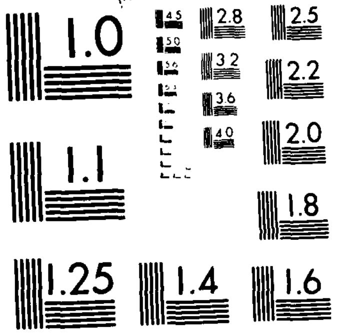
AD-A125 800 NATURAL ARTIFICIAL LANGUAGES: LOW-LEVEL PROCESSES(U) 1/2
CALIFORNIA UNIV SAN DIEGO LA JOLLA CENTER FOR HUMAN
INFORMATION PROCESSIN G PERLMAN DEC 82 ONR-8208
UNCLASSIFIED N00014-79-C-0323 F/G 5/7 NL


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6

(12)

ONR REPORT 8208
DECEMBER 1982

Gary Perlman

NATURAL ARTIFICIAL LANGUAGES: LOW-LEVEL PROCESSES

A

UNIVERSITY OF CALIFORNIA, SAN DIEGO

CENTER FOR HUMAN INFORMATION PROCESSING
LA JOLLA, CALIFORNIA 92093

The research reported here was conducted under Contract N00014-79-C-0323, NR 667-437 with the Personnel and Training Research Programs of the Office of Naval Research, and was sponsored by the Office of Naval Research and the Air Force Office of Scientific Research. The views and conclusions contained in this document are those of the author and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the sponsoring agencies. Approved for public release; distribution unlimited. Reproduction in whole or in part is permitted for any purpose of the United States Government.

Natural Artificial Languages: Low-Level Processes

Gary Perlman
Cognitive Science Laboratory
Center for Human Information Processing
University of California, San Diego

I wish to thank the UCSD Design Group: Steve Draper, Ben du Boulay, Jim Hollan, Ed Hutchins, Phil Mercurio, Don Norman, Dave Rumelhart, and Kathy Uyeda, and its extended group through network mail, notably, Bob Glushko and Mike Williams, who provided many stimulating discussions. Mark Wallen was instrumental in many of the programming aspects. Anne Sutter collected most of the data for the experiments.

This research was conducted under Contract N00014-79-C-0323, NR 667-437 with the Personnel and Training Research Programs of the Office of Naval Research, and was sponsered by the Office of Naval Research and the Air Force Office of Scientific Research.

Requests for reprints should be sent to Gary Perlman, Bell Telephone Laboratories, 5D-105, 600 Mountain Avenue, Murray Hill, New Jersey, 07974, USA.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER ONR-8208	2. GOVT ACCESSION NO. A125800	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Natural Artificial Languages: Low-Level Processes		5. TYPE OF REPORT & PERIOD COVERED Technical Report
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Gary Perlman		8. CONTRACT OR GRANT NUMBER(s) N00014-79-C-0323
9. PERFORMING ORGANIZATION NAME AND ADDRESS Center for Human Information Processing University of California, San Diego La Jolla, California 92093		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS NR 667-437
11. CONTROLLING OFFICE NAME AND ADDRESS Personnel and Training Research Programs Office of Naval Research (Code 442-PT) Arlington, Virginia 22217		12. REPORT DATE December 1982
		13. NUMBER OF PAGES 90
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distributtion unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

This research was also supported by the Air Force Office of Scientific Research.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Command Languages	Learning Mathematics
Computer Program User-Interface Design	Symbol Systems
Human-Machine Interaction	

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

OVER

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ABSTRACT

An artificial language is one created for concise and precise communication within a limited domain such as mathematics. A natural artificial language is one that people find easy to learn and use. I discuss low-level properties of natural artificial languages, especially those in which names are chosen for concepts, and symbols are chosen for names, a class of artificial languages I call linguistically mediated artificial languages. These properties include choosing mnemonic symbols for names, and suggestive names for concepts, and using both internally and externally consistent syntax. I outline a model of processing linguistically mediated artificial language and present results from experiments in support of the model. The results of the experiments are applied to the design of a user interface to a programming system, demonstrating their practicality along with their theoretical interest. The research shows the tradeoffs in designing natural artificial languages: naturalness in a specific domain is gained at the cost of generality for other domains.

INTRODUCTION TO BASIC CONCEPTS

In this paper, I explore languages for communicating precise ideas within limited domains, a class of languages I call artificial languages. Artificial languages differ from natural languages in that they are invented by people for communicating precisely within limited domains, and as such are superior to natural language for communicating ideas within those domains, and inferior for communication in domains for which they are not designed. Mathematical notation is an example of a general artificial language used to write expressions about mathematical relations. Artificial languages can be finely tuned to whatever domain they are intended. General purpose computer programming languages, such as ALGOL, are used to express general algorithms for computers. High level computer programming languages such as LISP or SNOBOL are even more specialized artificial languages.

Despite their importance in technical communication, there has been little experimental research on how to design artificial languages so they are easy for people to learn and use. It is my purpose to study how people use artificial languages. With a better understanding of this, artificial languages can be designed so they are easy to use. I describe some properties of artificial languages, in particular, properties I think make an artificial language a natural one for people to learn and use, and I outline a model of how people process artificial languages like mathematical notation and programming languages. In the framework of the model, I present experimental evidence in support of my claims that certain properties affect how easily people can use artificial language. Because I hope to be able to apply the findings of the research, I give examples of their application to the design of user-computer interfaces for computer programs.

In this section, I begin with an example of use of artificial language: writing a theorem of mathematics in mathematical notation. Then I introduce the terms this paper depends on heavily. I describe some desirable properties of artificial languages and then outline a theory of how people process artificial language. Then I go on to describe experiments on artificial language processing, followed by practical application of the results to the design of a user interface to a programming system. Finally, I offer conclusions about future possibilities for the study of artificial languages.

An Example of Artificial Language

An artificial language is often a shorthand for expressing ideas:

> By the aid of symbolism, we can make transitions in reasoning almost mechanically by the eye, which otherwise would call on higher faculties of the brain. By relieving the brain of all unnecessary work, a good notation sets it free to concentrate on more advanced problems. (Whitehead, 1911 (chapter 5)).

Artificial languages are used to express ideas with few symbols and in a small space, for fast and easy study. For many purposes, natural language is unacceptabl[illegible]rbose, and understanding the meaning of a statement becomes too much of a strain on short term memory. Many technical concepts are unintelligable when expressed in natural language because by the time a person gets to the end of a sentence, the content of the beginning is forgotten. On the other hand, by packing a lot of information into a small space, more information is available at a glance, and this can facilitate the observation of relations between concepts.

It is difficult to grasp the meaning of the following familiar theorem when natural language is used.

> In all triangles with an angle such that the sides forming the angle are perpendicular, the product of the length of the side opposite the angle such that the sides forming the angle are perpendicular with itself is equal to to sum of the products of the lengths of the other two sides, each with itself.

This version of the Pythagorean Theorem has several problems. It is too long, and this puts a load on the short term memory of any person reading it. Some of the descriptions, notably that of the hypotenuse, are so involved that it is hard to tell when they begin and end. In following paragraphs, I reduce this version to a more compact notation that is easier to understand. In doing so, I point out some of the issues important for designing an artificial language that is easy for people to learn and use.

By introducing new terms to represent complex ideas, and by introducing symbols for these and for operations used on the symbols, the theorem becomes shorter and more easily understood. To shorten the long version of the theorem, I first introduce some terms for the concepts that are important enough to warrant their own terms. Although there is a cost of introducing new notation because people have to learn it, assigning commonly denoted things their own names helps compact expressions. This process of naming concepts will be the focus of later sections.

First, a _right_ angle is one whose lines forming the angle are perpendicular. I choose this name because of _convention_ and not because I think it is a particularly good name--_square_ would be more suggestive--but _right_ is widely used. A _right_ _triangle_ is one with a right angle. (My preference above would yield the awkward term: _square triangle_.) A _hypotenuse_ is the side of a right triangle opposite the right angle. (Here, the following of traditional nomenclature is even more questionable.) To _square_ a quantity is to multiply that quantity by itself. With this bit of notation, the theorem becomes:

In a right triangle, the square of the length of its hypotenuse is equal to the sum of the squares of the lengths of the other two sides.

Now that names have been chosen for some important terms, the theorem can further be abbreviated by choosing symbols for names. In mathematics, single character symbols have traditionally been used, in part to be able to indicate multiplication by juxtaposition. I will use lower case letters to abbreviate the names of the triangle sides: hypotenuse (h), one other side arbitrarily chosen (a), and the other (b). By convention, the length of a side will be represented by the capitalized letter name for that side. I will use the symbol "=" to stand for "is equal to," the symbol "+" to stand for the addition operation, and the usual superscript numeral two to represent squaring. With these conventions, the theorem reads as:

In a right triangle,

$$H^2 = A^2 + B^2$$

Here symbols were used to make the Pythagorean Theorem a lot shorter than the natural language version abbreviated with several definitions. The question of how to choose symbols for names is another major theme of this paper.

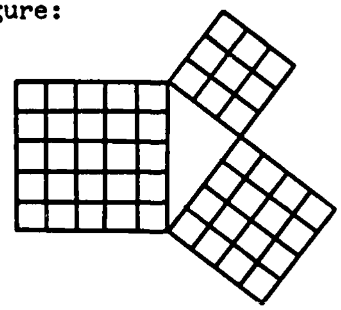
A Preview of Things to Come

The example of representing a concept with artificial language emphasizes several points I discuss throughout this paper. First, artificial language is used to abbreviate natural language. This speeds up the communication of ideas. Later, I argue that it also affects <u>how</u> people communicate. Second, there are different stages of converting natural language to artificial language. In the example, names were introduced to stand for commonly denoted terms, and then, symbols were used to abbreviate the names. Some of the names and symbols were chosen to be consistent with existing conventions. For the most part, the order of the symbols was kept consistent with the way the theorem is stated in natural language. Some of the choices of symbols were not guided by convention: I chose "h" to stand for "hypotenuse" because it is a natural abbreviation. While "h" is mnemonic for "hypotenuse," it has no relation to the concept of hypotenuse. Instead, it is an indirect mnemonic for the concept because it is mnemonic for a well known property of the concept, its name. This indirect mnemonicity is shared by all artificial languages that represent concepts with natural language names and abbreviate those names with symbols, a strategy I call <u>linguistic mediation</u>. Another strategy in representing concepts is for symbols to be designed to share properties of that which they represent.

Mathematical notation has symbols that can be manipulated abstractly. That is, the relationships between symbols and the concepts they represent are not based on inherent properties of the symbols themselves. Rather, the relationships are by definition. For example, in the equation:

$$H^2 = A^2 + B^2$$

there are no (or at most few) properties of the symbols that express the underlying concept without reference to an external set of definitions. Contrast the above with those symbols in mathematics that can be manipulated geometrically. With these, the symbols have physical properties that correspond to the properties of the concepts they represent, a property I call <u>analog</u> (also known as <u>iconicity</u>). The Pythagorean Theorem, represented abstractly above can be represented geometrically with a figure:



Here, the correspondence between notation and underlying concept is relatively high, and the concreteness of the example helps understanding many relationships not emphasized by the earlier more abstract equation. But note that the geometric representation does not generalize well. A problem with such analog systems of notation is that they emphasize some features at the expense of others. The example above is a good illustration of the concept of squaring the sides of a right triangle, but suggests that it only applies to triangles with sides in integer, or perhaps only 3:4:5 ratios.

The two versions of Pythagorus′ Theorem are at extremes of an abstract/analog continuum. Some linguistic constructs have mixtures of symbols with abstract and analog properties. For example, a hybrid of the two examples might be:

$$\boxed{H} = \boxed{A} + \boxed{B}$$

where the squaring operation is represented by placing a square around its operand. Note that while such a notational trick is useful here, it does not generalize well. Representing cubing would be difficult, and other powers would be impossible. Also, such a notation is inconsistent with existing notation. This brings up another major theme of this paper: that of tradeoffs between ideal solutions for specific applications and generalizability to other applications.

Natural Language vs. Artificial Language

There are other virtues to notation than mere abbreviation. An artificial language can shape the way people understand concepts written in that language. An artificial language introduces new terms that may affect how we comprehend the concepts written in that language. As Whorf (1956, p. 213) said, "We dissect nature along lines laid down by our native languages." I claim the same hold true for the way we understand concepts expressed in an artificial language. Artificial languages give us terms in which to express technical ideas, and the design of an artificial language affects the way we understand these ideas. High level computer programming languages are created because they provide primitives that are appropriate for a specific application. Pratt (1975) claims that a programming language provides a conceptual framework for thinking about algorithms and data structures as well as a means of expressing them. Some examples include LISP which supplies primitives for working with lists, COBOL for business applications, and APL for processing arrays and matricies. The tradeoff is that these artificial languages can be so tuned to one domain that they can not be easily used in any others. For example, APL would not make a good language for business applications because its primitives are weak for formatting and file handling. Thus, artificial languages in general, and programming languages in particular, can be fine tuned to the domains for which they will be used, but at the expense of applicability to other domains.

Currently, there is much debate about the usefulness of natural language for technical purposes. One argument is that if computers could understand natural language, then everyone would be writing programs and getting full use from computers. This has not been tested because of lack of such powerful systems, but historical evidence of the use of artificial languages suggests that people have used notation because they wanted to, not because they were forced to. Schneiderman (1980) suggests that natural language is an unnatural one for communication with computers. He claims experienced users prefer brief unambiguous forms and inexperienced users may not understand the limitations of computer systems and develop unrealistic expectations. Hill (1972) considers the possibilities after natural language systems are created.

> Would things really be any easier? The main difficulty in programming lies in deciding <u>exactly</u> the right thing to do. To put it into a programming language is relatively trivial.... And would we not throw away one of the greatest advantages of computers? Namely that they can be instructed to do exactly what you want without argument or misunderstanding.

Schneiderman (1980) agrees with this view and cites structured notations in mathematics, chemistry, and music used to help clarify problems. These languages developed over centuries because people needed terms with which to express precise ideas. Their existence suggests that artificial languages are not poor substitutes for natural language, but necessary means for expression.

David Taylor, visiting UCSD while on sabbatical leave from the University of Rochester, and I embarked on a study of how to design the "ideal" computer programming language. We began by observing people thinking aloud and writing down simple algorithms, such as for sorting words and string manipulation. We hoped we would discover the units of thought people preferred to use when not constrained by a particular language. Instead we found that people could not clearly state algorithms and tended to write down solutions to problems rather than provide a method for solving general cases. Even experienced programmers wrote unworkable natural language programs for algorithms they understood well. Without an artificial language as a constraint and a means of expressing thought, their thought lacked precision.

Given the importance in the past, and probably the future, of artificial languages in the communication of precise ideas, it is surprising that little psychological investigation has been done on how to design them. Clark and Clark (1977), a standard reference in the psychology of language, has no reference to any sort of notational systems. Semiotics, an area of linguistics studying the use of signs in language, including work dealing specifically with notational systems (Akhmanova, 1977), has little to offer in the way of predictive theories or experimental data on artificial languages. Finally, in the area of mathematics education, where interest in communicating technical mathematical concepts should be high, only lists of guidelines have appeared (Perlman, in press).

My goal here is to describe some properties of artificial languages that make them easy to learn and use, show experimental evidence that these properties do affect the efficiency of communication, and describe a model of artificial language processing that accounts for the effects of these properties. Because artificial languages are developed and used in technical domains, I also have the goal of being able to apply what I find about artificial languages to the design of new artificial languages.

Descriptions of Important Terms

Many of the terms I use are loosely defined in this section. Most of the definitions include examples for concreteness.

LANGUAGE: (a) Any means of expressing or communicating, as gestures, signs, animal sounds, etc. (b) A special set of symbols, numerals, rules, etc. used for the transmission of information, as in a computer. (Guralnik, 1978).

NATURAL LANGUAGE: A language occurring naturally in the world, usually having evolved over a long period of time. Examples: English, American Sign Language.

ARTIFICIAL LANGUAGE: A language created especially for precise and concise communication within a limited domain. Examples: Mathematical notation, computer programming languages, music notation, circuit diagrams.

ANALOG: The property that there exists a natural correspondence between the symbols in a language and the concepts they denote. That is, symbols have physical features that correspond to some features of the concepts they represent. Examples: In music notation, the placement of notes on the staff corresponds to their order and rhythm (moving from left to right) and to their pitch (moving up and down).

ABSTRACT: The property that the pairing between symbols and concepts is by convention alone and the symbols have little or no relationship with properties of the concepts they represent. Examples: In natural language, most of the names for things are arbitrary (though some names are onomatopoetic, e.g., "splash" sounds like what it denotes). In mathematical notation, many symbols are chosen as abbreviations for natural language names and as such, do not have much to do with the concepts they symbolize. Howe' 'r, some symbols are chosen to be mnemonics for English names. For example, we use the letter "i" to represent an <u>i</u>nteger. So while a symbol may have no intrinsic correspondence with the concept it symbolizes, it may have one with a well learned property of the concept, such as its name.

LINGUISTIC MEDIATION: The process of using natural language terms as an intermediate representation between a mental or conceptual representation and a symbolic representation in an artificial language. A name is chosen for a concept, and then a symbol is chosen for that name. The name is a linguistic mediator. This process is the area of artificial language processing that gets the most attention in this paper. Example: In using a programming language to compute a mean, a person might refer to the sum of all the numbers as "sum," and symbolize that quantity with the symbol "s."

NATURAL ARTIFICIAL LANGUAGE: An artificial language that people find easy to learn and use: a well designed artificial language. This paper is a report of investigations on what properties make an artificial language easy to learn and use, why, in psychological terms, these properties are helpful efficient communication, and how knowledge about these properties can be applied to the design of natural artificial languages, particularly to the design of user interfaces for computing systems.

Properties of Natural Artificial Languages

In this section I discuss some properties that make artificial languages easy to learn and use. This list is not meant to be exhaustive, and is based on linguistically mediated artificial languages. I refer to psychological theory as evidence for why these properties are desirable, and in later sections, I present experimental data in support of my claims.

Mnemonic Symbols

A <u>mnemonic</u> is an aid to memory. I claim that symbols should be chosen so that what they represent can most easily be recalled. Consider the famous equation:

$$E = mc^2$$

which states that the energy in matter is equivalent to its mass multiplied by the speed of light squared. Ignoring its deeper implications, the equation has five symbols in it, two of which, the equal sign and the squaring operation, are standard and will not be discussed. The capital letter 'E' is an abbreviation for the word "energy." As such, it is an aid to memory, because "E" is more closely associated with "energy" than other letters because it is the first letter of the word it represents. The same can be said for "m" representing "mass." The speed of light being represented with "c" is an example of a non-mnemonic choice of symbol. Perhaps "c" is the initial letter of a foreign term for speed or light (the Latin <u>celer</u> means "fast"). This possibility points out a problem with choosing initial letters as symbols to represent names: this linguistically mediated symbol depends on the natural language used. What is a mnemonic choice of symbol in one linguistically mediating language may be a foil to recall in another. Language neutral symbols can be chosen that are better than the worst case for some mediating languages, but they are not the most mnemonic that can be chosen for any one language. This tradeoff is shown in later experiments.

Suggestive Names

In a linguistically mediated artificial language, to understand some notation, it is necessary to associate symbols with the concepts they represent. The process involves activating the name of the concept via its symbol, and then the name is used to activate memory structures associated with that name. My hypothesis is that since the name activates memory structures, and these structures represent the understanding of the concept, the choice of name affects how a concept is interpreted, independent of the formal properties of the concept. Furthermore, the activated concept in turn activates related concepts and their names and symbols, which affect the processing of notation around it. In an artificial language, the name for a concept serves as a key to memory structures in terms of which concepts around it are interpreted.

There are tradeoffs in using suggestive names. By choosing a specific name over a general one, more information about the concept is conveyed, and this can give people more intuitions about the properties of the concept. This has the advantage that people are provided with a relatively rich set of knowledge structures to use to interpret incoming information (Rumelhart and Norman, 1978). However, some properties may be enhanced at the expense of others and a cognitive set can bias people away from novel interpretations of the concept being represented (Halasz and Moran, 1982).

Consider a problem in linear programming: to maximize a function subject to some set of linear inequalities. A program describing the algorithm to do this might use general names like makebig for the quantity to be maximized, and keepsmall as the name for the set of limits. This choice of names does not bias a person toward one application area over another, which is good for generalizing the knowledge to many domains. However, as one of the reported experiments shows, the general names make learning the algorithm more difficult for any one domain because the names are less suggestive of the relationships among concepts. A specialized business program might use names like profit and expenses, and this would improve learning for the business domain. Learning the business program biases people against considering novel uses of the program, such as to maximize nutrition while limiting calories or cost. This is another example of a tradeoff between an ideal solution in a limited area being less than ideal as a general solution.

Consistent Syntax

In an artificial language, the symbols are arranged according to some syntax that defines the role of the symbols in expressions. This syntax is a set of rules which must be easy to recall if efficient (fast and correct) communication is to be achieved. If there is a simple syntax for all linguistic constructions then it will be easier to learn and use than a collection of special cases. If similar concepts are expressed with similar syntax, then what people know about one part of an artificial language helps them guess how other parts are represented. This guessing involves applying analogical rules (Rumelhart and Norman, 1980) learned in one situation to others by making minor modifications to the old knowledge. For example, a program to move one file to another might look like:

```
move oldfile newfile
```

so that the movement of data is from left to right. An analogically consistent command to copy a file would be:

```
copy oldfile newfile
```

If the command for copying used a different syntax,

```
copy newfile oldfile
```

then a person generalizing from the move command to the copy command would be prone to make errors. Note that the second copy command could be analogically consistent with the common computer programming language assignment statement:

```
newfile = oldfile
```

The analogical consistency depends on how the rule governing the syntax is explained.

Of course, it is necessary to know the rules to be applied. Without knowledge of the syntactic rules, there can be no analogy; each construction is viewed as unique. The same language can be described by several grammars, each of different cognitive complexity. Reisner (1981) found this to be the case with languages for describing computer graphics. She found that the number of production rules in a grammar for identical languages was positively correlated with the difficulty people had in learning the languages. An interesting experience the people in our laboratory had is a good example. The text editor we use has an elegant underlying model for the commands that can be executed. Each command can be given an area of text to work on by supplying a leading list of lines, or a trailing context search. This allows similar commands for operating on characters, words, lines, sentences, paragraphs, etc. The commands can be deletion, change, yank into a buffer, and so on. The commands read out loud like "change word," "delete sentence," "move three lines," and so on. The interesting point is that many users do not know about this underlying model but gradually discover it. Once the rules for forming commands are known, it is then possible to infer how a command will work by changing a variable (either command or text area).

To say that syntax should be consistent is too simple minded. It does not make clear what is consistency. A syntax is <u>internally consistent</u> if its syntax follows without exceptions a small set of rules. LISP is a programming language with an internally consistent syntax: all functions calls are of the form

```
(NAME ARG1 ARG2 ARG3 ...)
```

NAME is the name of the function, and the ARG's are the arguments. To decide how a command should be written, only one rule need be used for the general form, though the number and order of arguments is often problematic.

A syntax is externally consistent if it follows some set of rules outside the artificial language, such as ones from an existing artificial language. LISP is not externally consistent in the way it implements calls to arithmetic functions. To write the formula:

$$A + B * C - E$$

one has to write in LISP:

```
(SUM A (DIFFERENCE (TIMES B C) E))
```

This follows its simple internally consistent syntax, but this is inconsistent with everyday arithmetic with which people are more familiar. Thus, consistency can only be measured relative to a particular set of conventions.

Internal consistency allows learning by analogy within the language, and external consistency allows transfer of existing knowledge. At initial stages of learning a language, external consistency may be important because the first language construct can be learned by analogy with those of known language. This helps a person gain a foothold in the new language. Later, as a person becomes more immersed in the language, internal consistency plays a more important role in ease of use. It turns out that all possible combinations of internal/external consistent/inconsistent languages exist, summarized as follows:

	INTERNAL	
	Consistent	Inconsistent
EXTERNAL		
Consistent	Existing good notation	Existing bad notation
Inconsistent	A historical problem: Transfer of knowledge is impossible or detrimental	Piecemeal notation "As is needed"

Maintaining Notation

There is no hope of replacing long established notation--the only hope of establishing a good notation is at the outset. (Skemp, 1971 (Chapter 5))

The adoption of notational conventions makes it easier for people to communicate without prefacing each communication with definitions. This takes time for defining terms and time for the receiver of information to learn them. However, when it comes time for a new notation to be designed, having to adhere to conventions can make the new notation a poor one. This is true for choosing symbols for names, for choosing names for concepts, and for defining a syntax for the language.

In the choice of symbols: In the area of theoretical computer science called automata theory, a Turing Machine is traditionally denoted by a septuple:

$$M = (Q, \text{SIGMA}, \text{GAMMA}, \text{delta}, q_0, B, F)$$

where

Q is the finite set of states,
GAMMA is the finite set of allowable tape symbols,
B, a symbol of GAMMA, is the blank,
SIGMA, a subset of GAMMA, not including B,
    is the set of input symbols,
delta is the next move function
q_0 in Q is the start state
F is a subset of Q of final states.

This example is the standard notation used, and is taken from Hopcroft and Ullman (1979). Some of the symbols are mnemonics: "B" for "blank", "F" for "final states," and even q_0 is mnemonic relative to Q, but the others seem to be chosen randomly. It is not surprising that some writers have tried to introduce new notation (Savitch, 1979) though it is difficult to overcome the inertia of a well established set of conventions, no matter how poorly they were motivated.

In the choice of names: We use the word "editor" to denote a computer program we use to write papers, but in fact the program does no editing and misleads people about its purpose. Our editor programs have a command called "print," used to display specified lines of an edited file on its users terminal. Bott (1979) found that names like "print" were misinterpreted by novice users because the name has many associations independent of the formal properties of the command. For example, the term "print" tended to elicit associations to concepts related to writing block letters, and not to the less familiar computer concept.

In the choice of syntax: Mathematical notation has evolved over centuries, but there are examples of inconsistencies of syntax. Arithmetic usually has operators between operands, transcendental functions usually precede their parenthesized operands, and many matrix operations follow theirs. It is therefore impossible for an artificial language to be internally consistent without breaking long standing tradition.

Existing notation, however poorly designed, imposes restrictions on the form of new notation. In designing an artificial language, there are tradeoffs between having a notation consistent with existing languages and one that has the best choices for symbols, names, and syntax. Adhering to conventions is not the only source of tradeoffs. Later, I show that there is no such thing as an ideal natural artificial language, only tradeoffs in design. I show in a series of experiments that the more that is known about the domain of application of an artificial language, the more finely that language can be tuned to the domain, but at the expense of applicability to novel purposes.

A Model of Artificial Language Processing

The model I outline in this section applies to processing linguistically mediated artificial languages. Recall that in a linguistically mediated artificial language, names are first chosen for concepts, and then these names are abbreviated by symbols. This is represented diagramatically as:

symbols <==> names <==> concepts

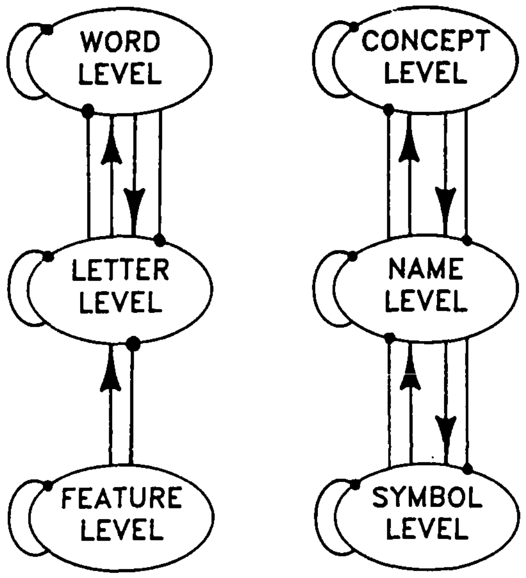
The arrows are bidirectional to stress the model applies to the comprehension of notation (left to right) and to the generation of notation (right to left). The model I present is an <u>interactive</u> model in that processing does not move in one direction, such as from symbols to names to concepts during comprehension. Rather, as processing moves from lower to higher levels, the higher levels feed back expectations to lower levels. This is explained further below.

The model is based on the McClelland and Rumelhart (1981) interactive model of word and letter perception. Before discussing my model, I summarize theirs. The McClelland-Rumelhart model is used to explain, among other phenomena, the "letter-word" superiority effect: letters are easier to recognize when they appear in words than in isolation. They explain this by a multi-level interactive activation model. The levels in the model are for representations of <u>features</u>, <u>letters</u>, and <u>words</u>, and are summarized in Figure 1a.

> In the model, perception results from exitatory and inhibitory interactions of detectors for visual features, letters, and words. A visual input excites detectors for visual features in the display. These excite detectors for letters consistent with the active features. The letter detectors in turn excite detectors for consistent words. Active word detectors mutually inhibit each other and send feedback to the letter level, strengthening activation and hence perceptibility of their constituent letters. (McClelland and Rumelhart, 1981 (abstract))

Perception begins at the feature level with input from some stimulus. Features that are present in the stimulus become activated; their activity levels increase, relative to a neutral state without

Figure 1: Interactive Models of Reading and Symbolism



Shown here are diagrams of the McClelland-Rumelhart (1981) model of word perception, and my model, based on it, of processing symbolic notation. In part (a) on the left, three levels of processing are depicted: the feature, letter, and word levels, connected with excitatory (arrows) and inhibitory connections (arcs ending in dots). In part (b) on the right, the model is extended to include a conceptual level. The name level roughly corresponds to the word level in part (a) while the symbol level in (b) roughly corresponds to the letter level in (a).

simulation. Features that are part of letters activate those letters. For example, a "vertical bar" feature activates letters like "b" or "k" but not letters like "s" or "o" which are activated by "curving line" features. Additionally, McClelland and Rumelhart assume inhibitory connections between features and letters in which they do not appear, and between any two elements of any level.

After activation spreads from the feature level to the letter level, it continues to the word level. By a similar process as between features and letters, letters activate words in which they appear, and inhibit the activation of words in which they do not. Once words are active, the interactive part of the model takes place. Activated words send expectatory feedback, or "top-down" activation to lower levels by activating their component letters. For example, having the letters "s," "n," and "k" active increase the activation of words like "sink," "sank", and "sunk," and these words being active enhance the perceptibility of letters like "i," "a," and "u," but not other letters, such as consonants.

My model of processing linguistically mediated artificial language has a similar multi-level structure and a similar interactive feedback process. In the model, there are three levels: the <u>symbol</u>, <u>name</u>, and the <u>concept</u> levels, shown in Figure 1b. Roughly speaking, the McClelland-Rumelhart model's letter level corresponds to my model's symbol level, and their model's word level corresponds to my model's name level, but this could be a misleading analogy.

In my model, the comprehension of notation follows a process similar to that modeled by McClelland and Rumelhart. Processing begins at the symbol level. Symbols activate the names they represent to the extent they are well chosen symbols. What determines "well chosen" is the topic of investigation of later experiments. Well chosen, or <u>compatible</u>, symbols positively activate the names they represent, while poorly chosen symbols can inhibit the activation of the names they represent, or equivalently, activate a competing name. Activated names in turn activate the concepts associated with that name, which depend on a person's experience with a name. The process involves the initial priming of concepts by their names, followed by the spreading activation among concepts not unlike that described by Collins and Loftus (1975). Activation spreads to related concepts--which again depends on world knowledge--which activates the names associated with them. This "top-down" feedback activation continues to aid the processing of compatible symbols for the activated names, notably letters beginning the names. This interactive part of the model is used to predict how the choice of name for a concept can affect how surrounding notation is processed.

I conclude the discussion of the model of artificial language processing with an example that may clarify the basic process. I hope to illustrate the way symbols have associations to the names they abbreviate, and how this association can depend on the particular names chosen for concepts. This example is drawn from the use of a program written by me called DESC, that <u>desc</u>ribes a single distribution of data (Perlman, 1980). DESC calculates summary statistics and prints frequency tables and histograms. To choose what output DESC produces, users append single character option letters after the program call of DESC. To get a histogram, an H is appended to the program. To get order statistics, an O is appended, and so on. Some options can be used to control the format of other options; the interval width (width of bins) of histograms can be specified by appending the letter I followed by the desired interval width. These symbols can be appended in any order. A fairly complicated call is:

```
DESC F H I10 M0 O T75
```

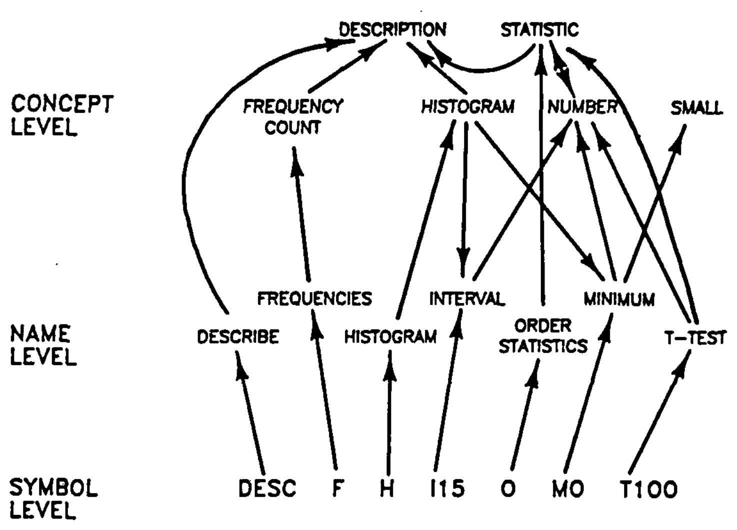
(I am using capital letters to make the symbols easier to read in the text; in actual use, they would all be lower case.) From experience, this compact notation is learned relatively easily by regular users. The following scenario assumes a user already knows what DESC is used for. First, the user reads the name of the program, DESC, and this activates concepts related to describing a single distribution of data: summary statistics, frequency tables, etc. A more suggestive name might be HIST, which would more highly activate concepts specifically related to histograms, but perhaps at the expense of concepts related to summary statistics. Then the option symbols are processed: F is an abbreviation of FREQUENCY TABLE, H is an abbreviation of HISTOGRAM. With this in mind, some concepts related to histograms and frequency tables are activated, and help interpret that: I is an abbreviation for INTERVAL WIDTH, and M is an abbreviation for the minimum value of the histogram and frequency table. Also, the I and M options are accompanied by values setting the interval width and minimum respectively. This can help interpret the numbers as the values, but also, the presence of the numbers can aid the comprehension process that I is short for INTERVAL WIDTH and M is short for MINIMUM. This is to stress that the processing is not strictly linear left to right, but involves the simultaneous activation of many concepts related to the tokens being processed. The rest of the options are O (for ORDER STATISTICS) and T75 (for T-TEST with Ho: Mean=75). Combined, the command suggests some sort of classroom test because the scores are being plotted greater than zero with intervals of ten units with the mean around 75. A command like:

```
DESC F H I15 M0 O T100
```

might be used to look at a distribution of IQ scores. Thus, concepts of even higher levels than general descriptive statistics can be helpful in understanding such low level artificial language constructs. A diagram showing some of the facilitory associations between tokens in the program call of DESC is shown in Figure 2. The notation here follows Norman and Rumelhart's (1977) scheme for representing semantic networks, and McClelland and Rumelhart's (1981) notation for neural networks.

The interactive model of processing linguistically mediated artificial language provides a framework for the experiments I report. In Experiment 1, I look for evidence of the facilitory and inhibitory activations from symbols to names that occur during comprehension of notation. In Experiment 2, I look for similar evidence, but for the task of generating notation. Together, these experiments suggest how to choose symbols for names. In Experiment 3, I look for evidence that conceptual knowledge, primed by the name given to the concept, can

Figure 2: Some Associations in a Call to DESC



Shown here are a few of the associations between concepts, names, and symbols in an example call to the descriptive statistics program DESC. At the lowest level are the actual symbols used in the call. Above them are the names the symbols represent. Finally, some associations among the concepts are shown. Note how the processing of the H option activates the name and concept of histograms, which in turn, activates the subsidiary concepts of interval width and minimum, which guide the processing of the I and M options, respectively.

affect how surrounding notation is processed. This last experiment provides some support for the notion that in a natural artificial language, the <u>form</u> of expression helps convey the <u>content</u>. After presenting these experiments, I show how their results can be applied to the design of an artificial language. Specifically, I discuss their application to the design of a user interface to a programming system. In the final section, I suggest directions for future research on artificial languages, and its potential for application in practical settings. I point out the tradeoffs between maximizing the effectiveness of an artificial language for a particular domain and one that can be used in a variety of domains. One onclusion is that the more one knows about what is to be represented, the better an artificial language for it can be designed. On the other hand, if an artificial language is to be used for many purposes, design decisions have to be made at a lower common denominator, to insure that new applications are easier with a more general language.

EXPERIMENTAL EVIDENCE

SYMBOLS <==> NAMES

In this section, I discuss experiments on optimal relations of symbols to names to answer the question: What is the best symbol for a name? I first address the question of how to choose symbols so that given a symbol, the name it represents can be most easily recalled. This is of primary importance to comprehending expressions written in a linguistically mediated artificial language. Then I address the question of how to choose symbols so that given a name, the symbol that represents it can most easily be recalled. This is important for generating notation. Finally, I discuss the applied implications of the studies for the design of menu based user interfaces to computer programs.

The general paradigm used is a simple one. I present people with a list of names paired with symbols. The task is to learn the paired association between these. In the comprehension task of Experiment 1, an experimental trial consists of the presentation of a symbol, followed by the subject <u>speaking</u> the name paired with it. In the generation task of Experiment 2, a trial consists of the presentation of a name, followed by the subject <u>typing</u> the symbol paired with it. In each case, the responses are timed and scored for accuracy.

One difference between the experimental tasks here and most traditional paired-associate learning tasks is that the list of symbol-name pairs is present while people are learning it. The main reason for this is to maximize practical applicability. In Experiment 2, I study optimal name-symbol pairings for the task of generating notation: Given a name, generate its symbol. This is a common task in many computer system user interfaces in which options are presented to users in a menu from which they can make selections. Selection is often made by pressing a key paired with the option. This is equivalent to generating a symbol for a name. I later discuss the application of the results of the experiments to such a menu based user interface. Because I am interested in being able to apply the findings in these experiments to the design of artificial languages, user interfaces among them, I have tried to make the experimental conditions as close as possible to some applied setting, without sacrificing the applicability of the results for artificial languages in general.

I limit the investigation to single character symbols for two reasons. Historically, they are the most widely used in mathematics. More recently, they have been used in computer systems to indicate desired selection from menu displays. Multiple character abbreviations have the disadvantage that people using them have to learn several characters, and often have to learn abbreviations of names (a type of symbol) that vary in length, making it a problem to know when the symbol ends. They have the advantage that they are richer in content than single character abbreviations. Akhmanova (1977) provides a list of the abbreviation strategies available. Previous experimental research on constructing symbolic abbreviations include studies of abbreviating by truncation (dropping trailing letters of a word) and contraction (removing certain classes of letters (e.g., vowels) (Hodge & Pennington, 1973; Ehrenreich & Moses, 1981). These studies found that any truncation governed by a rule was better than ones without (Reber, 1969); the latter can occur when different people use different rules. Other research has had people generate their own abbreviations (Streeter et al, 1981). A more recent expansion and refinement of previous research is that of Hirsh-Pasek et al (1982), who studied the learnability, encodability, and decodability, of names in a limited lexicon. They found that simple truncation is the most learnable (least trials to learning criterion) and encodable (easiest to generate abbreviation from name), but that a phonological system was superior for decodability (generate the name given the abbreviation). Underwood (1964) studied the effects of meaningfulness of terms in paired-associate learning and found that systematically related pairs of letters were more quickly learned and resulted in the best retention. Unfortunately, none of these studies looked at the time course of the association between names and symbols, information critical to bear on the framework based on the McClelland-Rumelhart (1981) interactive model of letter and word perception.

A Motivating Example: Symbols <==> Names

Before presenting the experimental evidence, I begin with an example that illustrates the results of these experiments. This example could well be drawn from a mathematics textbook on integer programming for high school.

> John wanted to buy some food but was short on money. With two dollars in cash he went to a grocery store where bananas and apples were on sale; apples were 30 cents each and bananas were a quarter. Assuming John liked apples one and a half times as much as bananas, how many of each should he buy?

There are several ways to find the optimal solution of five apples and two bananas. Suppose we begin by choosing variable names for the quantities. Let A be the cash John has on hand, B, the number of apples John buys, and C, the number of bananas. The problem is represented as:

```
Maximize John's happiness = 1.5B + C
Maintaining the restriction: .30B + .25C <= A
```

The symbols here are purposely chosen to be about as bad as possible given the simplicity of the problem. B stands for apples, not bananas.

A doesn't symbolize apples but cost, and C is used for bananas instead of cost for which it would be a better mnemonic. I claim that people would have more trouble with this problem with these choices of symbols than the obvious: A for apples, B for bananas, C for cost. I further claim that the problems would be less even if symbols unrelated to the names here were used. That is, instead of using A, B, and C, which can be confusing because they begin the names of quantities they do not represent, X, Y, and Z would be preferable because the pairing is neutral. While this is a loaded example, the situation is not far from what can happen when symbols for names are chosen without any consideration. In this section, I present experimental results showing the consequences of such poor design.

Experiment 1: Symbols ==> Names

In this experiment, I investigate the nature of the associability of symbols to the names they represent. The main concept introduced in this experiment is that of compatibility. A symbol is compatible with a name if the name is efficiently retrieved from the symbol, generally because there exists some mnemonic rule relating them. Efficiency includes both speed and accuracy. The general paradigm used to study this involves presenting people with a list of names, choosing some set of symbols for those names, and having the people speak the names as their symbols are presented. Four main conditions are compared: compatible letter symbols, incompatible letter symbols, compatible number symbols, and incompatible number symbols. The expectation is that compatible letters will be the easiest to associate with their paired names while incompatible letters the most difficult.

Method

Materials

Four conditions determined the stimulus sets of symbol-name pairs. Two stimulus lists of eight concrete nouns were randomly selected so that the eight words began with the first eight letters of the alphabet. Concrete nouns were used because Paivio (1969) has shown that ratings of concreteness, and the ease with which a word evokes a mental image, are good indicators of ease of learning. Quantities of nouns are also the most commonly abbreviated variables in mathematical notation, to which I want the results of the study to apply.

ant	apple
book	bear
cup	crane
desk	door
edge	exit
fiddle	flag
gully	gun
hawk	hurdle

There were two types of symbols: letters and numbers. There were two compatible sets for which there was a clear mnemonic rule relating the symbols. For compatible letter symbols, the symbol was chosen as the first letter of the name it represented. For compatible number symbols, the symbol was chosen to be the number corresponding to the ordinal alphabetical position of the first letter of the name it represented. For example, "5" represented "edge," and "3" represented "crane." There were two incompatible sets of symbol-name pairs. Incompatible letters symbols were randomly paired with names with the restriction that no symbol was the first letter of the name it represented. Incompatible number symbols were paired with names so that the number paired with a name corresponded to the ordinal alphabetical position of the randomly selected letter symbol. In other words, the random order of the names was the same for both incompatible symbol conditions. This was meant as a control condition for incompatible letters to compensate for list searching strategies, so one list would not be easier to search than another. It was anticipated that subjects would be less able to recall the pairings between symbols and names in the number symbol and incompatible letter symbol conditions, and would have to search through the list of symbol-name pairs when they could not remember the pairing. The four stimulus set pairs were sorted so that the symbols, the part of the display subjects would have to search, were always in alphabetical or numerical order.

Procedure

Subjects were seated in front of a micro-computer. A voice-key and a microphone were connected to the computer to determine when subjects made their responses. A tape recording of the session was made to record their responses for post-experimental error analysis. The order of presentation of the four conditions was arranged in a Latin square to counter-balance order effects among subjects. At the beginning of each of the four blocks of trials the list of symbols and names for a condition was displayed in a table in the upper left corner of the computer's screen. This list remained present for the duration of trials for that condition. On each trial, a symbol appeared in the lower right quadrant of the screen, and the subjects' task was to speak the name paired with the symbol as quickly and accurately as possible. Subjects were presented 10 random permutations of the symbols in each condition at about three second intervals, depending on their response speed.

Subjects

Sixteen University of California San Diego undergraduates received course credit for participating in the experiment.

Results

I first present the main results of the experiment, including the effects of compatibility and symbol type on the speed with which people could associate names to symbols. Then I present the results of practice effects to see changes in strength of any effects over time. I show how subjects' search strategies of lists can be determined from different reaction times for different list positions. Finally, I discuss the error analysis of the data. There were no significant differences between the two stimulus sets, nor were there any interactions of stimulus set with any factors, so to simplify the analyses, the data were pooled over that factor.

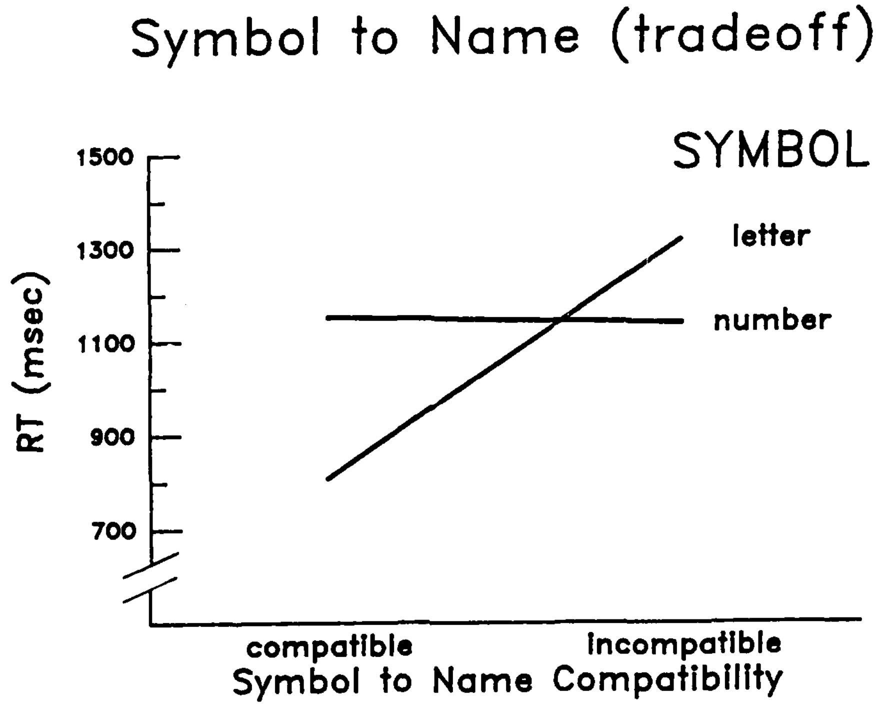
Major Results

The results in Figure 3 show the interaction (_F_(1,15) = 155.3, _p_ < .001), between compatibility and symbol type. The mean reaction times for speaking names paired with symbols are shown below:

Compatibility	Symbol	Mean RT
compatible	letter	808
	number	1152
incompatible	letter	1318
	number	1138

The data show that compatibility has little effect on number symbols, and a large effect on letter symbols, probably due to interfering activations of the first letter of a name and its symbol. See the later presentation of error data. A 99 percent Scheffe confidence interval indicated that compatible letters were significantly better than compatible numbers, and incompatible letters were significantly worse than incompatible numbers. The compatible letter case was the fastest condition, so much so that it helped produce overall advantages for letters over numbers (_F_(1,15) = 12.6, _p_ < .01), and for compatible symbols over incompatible (_F_(1,15) = 133.1, _p_ < .001).

Figure 3: Tradeoff Between Compatible and Incompatible Letter Symbols



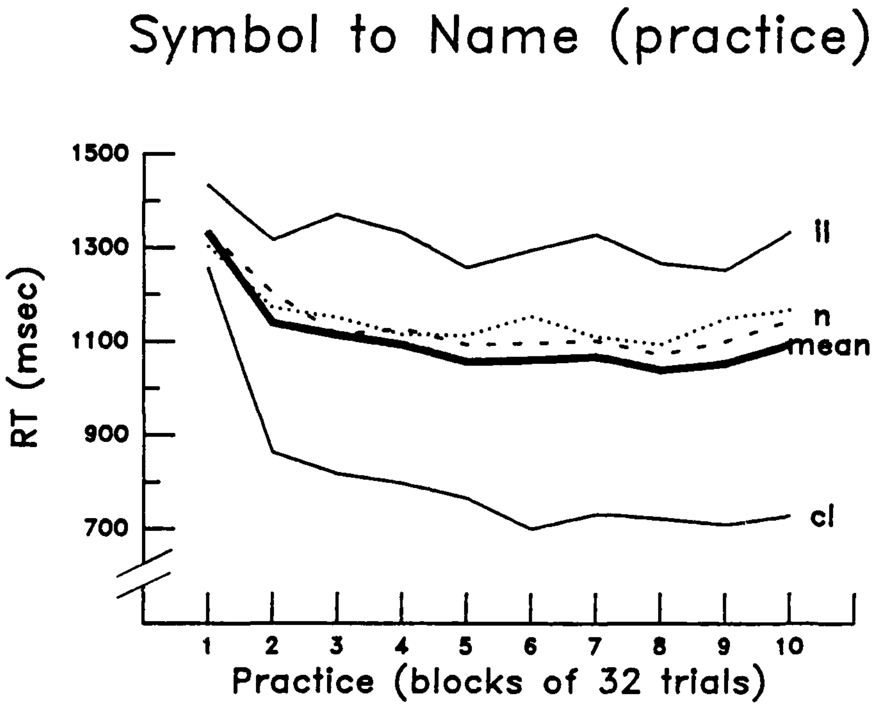
Shown here are the main results of Experiment 1, the mean times to speak the name of a symbol when presented with the symbol. Compatible letter symbols were the easiest symbols to learn (significantly better than compatible numbers) while incompatible letters were the worst (significantly worse than incompatible number symbols).

It is important to realize that while compatible letter symbols were by far the best, incompatible letter symbols were significantly the worst. This means that any effects favoring compatible letters can not be attributed solely to not having to search the list of symbol-name pairs for the correct answer. Consider the process subjects might go through when in the experiment. On presentation of the stimulus symbol, subjects had to recall the name paired with it. If they could not do so, they would have to refer to the table of symbol-name pairs in the upper left corner of the screen. This list searching process takes time and shows up in the overall reaction times for conditions for which list searching is necessary. Later, I present data that indicates subjects were more able to memorize the pairings in the compatible letter condition than in the other three conditions by showing no serial list position effects for compatible letter pairings, indicating subjects did not have to search the lists as often. Facilitory effects of compatible letter symbols might be attributable to not having to search the list of symbol-name pairs, but inhibitory effects of incompatible letters are directly comparable to the number symbol conditions because there is evidence of list searching in both. See the later discussion of list position effects.

Practice Effects

An important question is whether the effects shown in Figure 3 are stable over time, or due to an initial confusion about the task. Of course, subjects got faster with practice ($\underline{F}(9,135) = 19.0$, $\underline{p} < .001$), though toward the end of each condition, they showed signs of fatigue with reaction times increasing slightly. In Figure 4 it is clear that the advantage for compatible letters (cl) and disadvantage for incompatible letters (il) compared to the two number symbol conditions (n) are stable throughout the experiment consisting of 320 trials, or 80 trials per condition. The mean reaction times for ten blocks of trials, averaged over conditions, is shown as the bold line. The advantage for consistent letters only took effect after the first block of trials

Figure 4: Stability of Symbol to Name Compatibility



Shown here are the mean response times for the four conditions of Experiment 1, with the overall average practice curve shown as the bold line labelled "mean." Note that the differences among conditions do not diminish with practice. Compatible letters (cl) are always superior to other conditions while incompatible letters (il) are always inferior. The two dotted lines labelled "n" represent the virtually indistinguishable number symbol conditions.

($\underline{F}$(9,135) = 4.4, $\underline{p} < .001$); perhaps subjects were becoming familiar with the display and it took time to realize the symbol-name relation. This explains why subjects got better faster in the letter condition than with number symbols ($\underline{F}$(9,135) = 2.3, $\underline{p} < .02$), and better faster with consistent symbols than inconsistent ($\underline{F}$(9,135) = 2.2, $\underline{p} < .03$).

List Position Effects

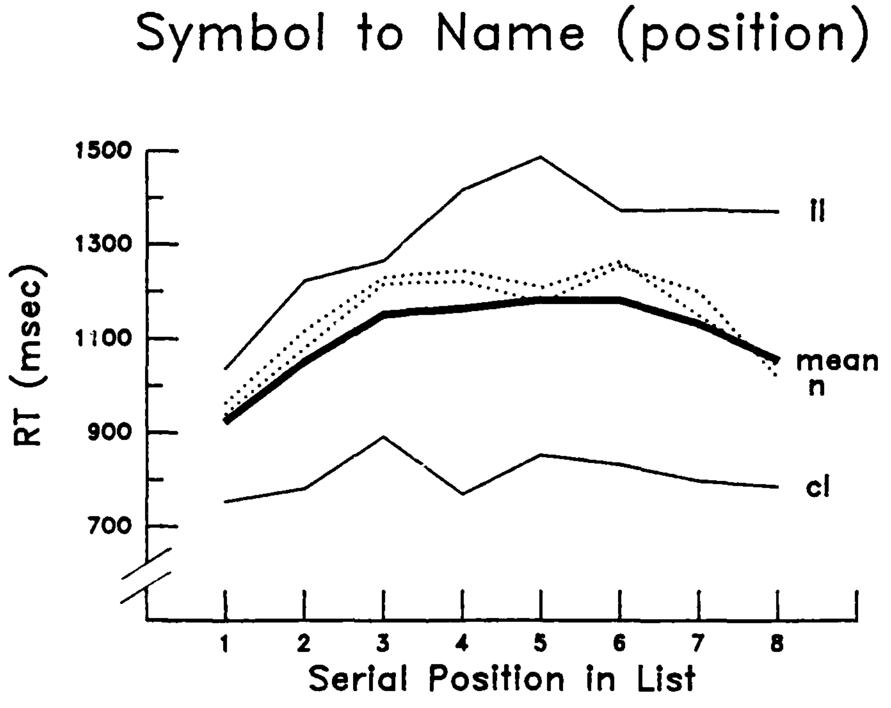
In Figure 5 the mean reaction times according to list position are shown with the bold line being the average over all conditions. The interaction between list position and the main experimental factors was significant ($\underline{F}$(7,105) = 30.4, $\underline{p} < .001$). In general, subjects were faster at finding names at the ends of the lists, with a superiority for the beginning of the list. This was true for all conditions except the compatible letter condition. With compatible letter symbols subjects probably did not have to search through the list as often as the other conditions, and so had about the same reaction times for all serial list positions. This resulted in special superiority for letter over number symbols ($\underline{F}$(7,105) = 5.7, $\underline{p} < .001$), and for compatible symbols over incompatible, ($\underline{F}$(7,105) = 5.0, $\underline{p} < .001$), and for compatible letters over other conditions ($\underline{F}$(7,105) = 5.2, $\underline{p} < .001$).

Errors

Subjects made few errors, probably due to pre-experimental instructions to be accurate. A response was scored as an error if the spoken response was not clear and crisp, or if the wrong response was made. Conditions where subjects made no response (eight in all) were dropped from any error analyses. The error rates for the four main conditions were:

Compatibility	Symbol	Errors	Percent	Mean RT
compatible	letter	17	1.3	1682
	number	29	2.3	1848
incompatible	letter	49	3.8	1886
	number	46	3.6	1945

Figure 5: Effects of List Position on Response Times



Shown here are the mean response times for each of the four conditions of Experiment 1 broken down by ordinal position in the list of symbol-name pairs. The average over all conditions is shown as a bold line labelled "mean." Note how there is an advantage for the ends of the list, especially for the beginning. This suggests the sort of search strategy subjects use: start searches for symbols at the ends of the list. Note also that there appears to be less of a serial position effect for compatible letter symbols (cl), apparently because the subjects did not have to search the lists in this condition as often as the others.

Compatible letter symbols resulted in about half the number of errors than any other condition but this trend is of marginal statistical significance (_F_(1,15) = 5.62, _p_=.03). Of possible note is that over one quarter of incompatible letter symbol errors, were the sort of errors predicted by competing activations (e.g., the stimulus "h" is paired with "gun," and the subject says the word in the list beginning with "h"). This is twice the proportion expected by chance with an eight item list, but this was not statistically reliable. Most of the errors in the experiment were attributable to speaking too softly for the microphone and voice-key to pick up their responses, or to stuttering. Errors were not due to a speed-accuracy tradeoff; the mean reaction times for errors in all main conditions were close to twice that of correct conditions.

Discussion

The results are straightforward. Choosing a symbol for a name is not to be left to chance, because a poorly chosen symbol for a name causes persistent problems in recalling the names they symbolize. The inhibitory effects of incompatible letters did not diminish over 80 trials, ten trials per symbol-name pair. These problems may be due to what Norman (1981) called _competing activations_. The first letter of a name is activated by the presentation of it as a symbol activates names that begin with that letter and inhibits the activation of names not beginning with that letter. This is analogous to letters activating words consistent with them in the McClelland and Rumelhart (1981) model of word perception. In addition, some experiments described by Rumelhart and McClelland (1982) showed context enhancements to word perception when a letter of a word is shown for brief periods (in the milliseconds) before presentation of the whole word, with the largest enhancement effect occurring with the initial letter of words. When the symbol is not the first letter of the name it represents, the loss is partly due to the lack of facilitory activation from the letter to the name it begins. Incompatible letter symbols are even inferior to number symbols because they cause the activation of other names, and the inhibition of the appropriate names. Thus, the symbols′ activated names, learned over a period of many years, compete with the loosely associated names they symbolizes. Number symbols, having few associations with names, do not cause facilitory or inhibitory activations.

This experiment tells us about comprehension of symbols, how easily a name can be recalled given its symbol. Now I turn to generation of notation. Given a name to symbolize, what is the easiest symbol to use so that abbreviation is efficient?

Experiment 2: Names ==> Symbols

In this experiment, I investigate the nature of the associability of names to the symbols used to represent them. In Experiment 1, I showed that compatible abbreviations of names serve as the best symbols in terms of efficiently recalling the name given the symbol. Here, instead of studying how people comprehend symbols, I study the generation of notation: Given a name to symbolize, what is the best choice of symbol so that symbol is most easily recalled? The approach is largely the same as in Experiment 1. People are shown a list of words paired with symbols much as they were paired in Experiment 1. The difference is that in Experiment 1, each trial of the experiment consisted of the presentation of a symbol and the subject had to speak the name it was paired with. In Experiment 2, on each trial, the name is presented and the subject has to type the symbol on a computer terminal keyboard.

This is a task that has become a common one. Many computer programs present options to people by showing them a list, or menu, of options from which they make their selections. Most programs do this by having the program user type a number or letter displayed beside the desired option. One simple strategy is to number the options so the first is selected with the number "1," the second with "2," and so on. Another scheme is to use letters of the alphabet, "a" for the first entry, "b" for the next, and so on. This scheme has the advantage of providing more symbols than numbers (26 compared to ten for single character symbols). Which is the best strategy? Experiment 2 is used to answer both the theoretical question of optimal schemes for generating symbols for names, and the applied question of optimal menu option selectors.

Method

Materials

One stimulus set of eight computer terms was selected so that each word began with a unique first letter of the alphabet between "a" and "h" inclusive. Below is the list of words, paired with consistent number symbols.

1	assemble
2	buffer
3	compile
4	debug
5	edit
6	file
7	graph
8	halt

The words were the same for all subjects, though their order changed according to conditions. The symbols were numbers (1-8) half the time or letters (a-h) the other half.

The experimental conditions were as follows. In half the conditions, the words were alphabetically ordered (sorted), and in the other half of the cases, they were randomly permuted. In half the conditions the symbols were alphabetically or numerically ordered -- in half they were randomly permuted. When both the words and symbols were permuted, the pairing between the two was the same as when both were sorted. This is a control condition. Conditions when both symbols and words are sorted, and when they are both permuted but their pairing the same, are called compatible. When targets are sorted and symbols random, and when targets are random and symbols sorted, the condition is called incompatible. In summary, there were eight experimental conditions: two symbol types (letter and number), two orders for the words in the list (sorted or random), and two orders for symbols (sorted or random). When both words and symbols were sorted, and when both words and symbols were randomly permuted, maintaining the same symbol-name pairings, the pairing was called compatible, and otherwise, it was called incompatible.

Procedure

The subjects' task was to type the symbol paired with a word in the list as quickly and accurately as possible after the word was presented. A microcomputer controlled the presentation of stimuli and the collection of reaction times with a standard CRT computer terminal. At the beginning of each condition, the list of symbols and names was presented in the upper left corner of the terminal screen. Subjects were allowed to study the display for as long as they liked, and the preview time was recorded without their knowledge. The list of symbols and names was present for all trials during each condition. For each condition, subjects went through five sets of eight trials, each set presenting all eight words in a different random order such that the same word was never presented twice in a row. Words were presented in the list in the lower right corner of the terminal screen at about five second intervals, depending on the speed of subjects' responses. When stimulus words were presented on the screen, they remained visible until subjects pressed a key. Subjects were instructed to press the symbol key with their preferred index finger, as quickly and accurately as possible while keeping errors to a minimum; accuracy was stressed more than speed. To compensate for different distances between keys, subjects were instructed to retract their hands to a point roughly equidistant to all keys on the keyboard between trials.

Subjects

Sixteen computer-naive University of California San Diego undergraduates received course credit for participating. Though the stimuli were computer terms, subjects did not have to have any knowledge of their meaning. Pilot data for the experiment indicated that subjects performed the same with computer terms as with the nouns in Experiment 1.

Results

Major Results

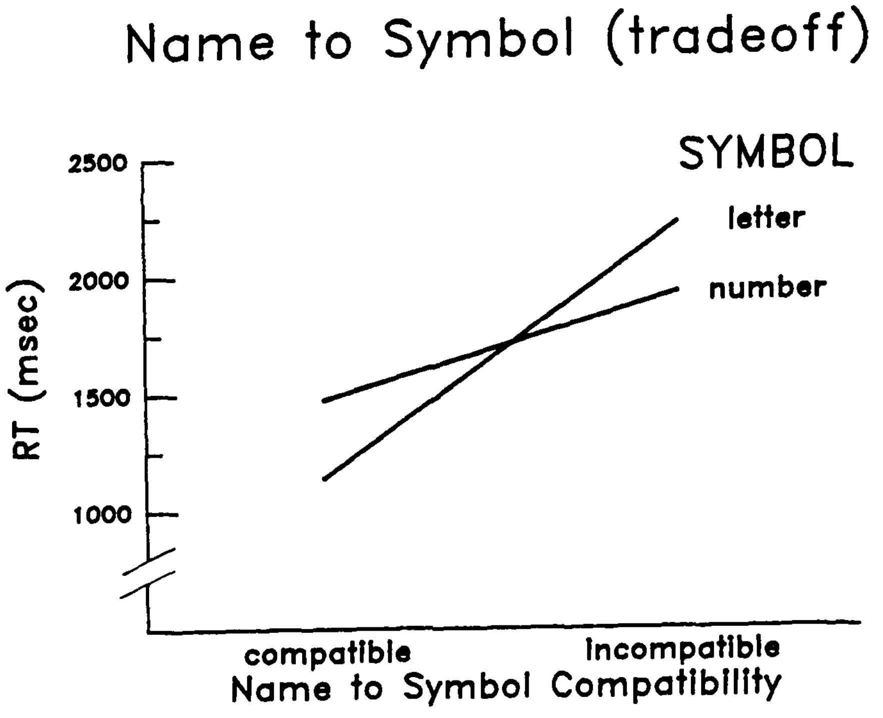
The data for the two factors with the most important effects are shown in Figure 6 and in the table below.

Mean Response Times (msec)

	Name-Symbol Compatibility	
	compatible	incompatible
Symbol Type		
letters	1139	2224
numbers	1474	1928

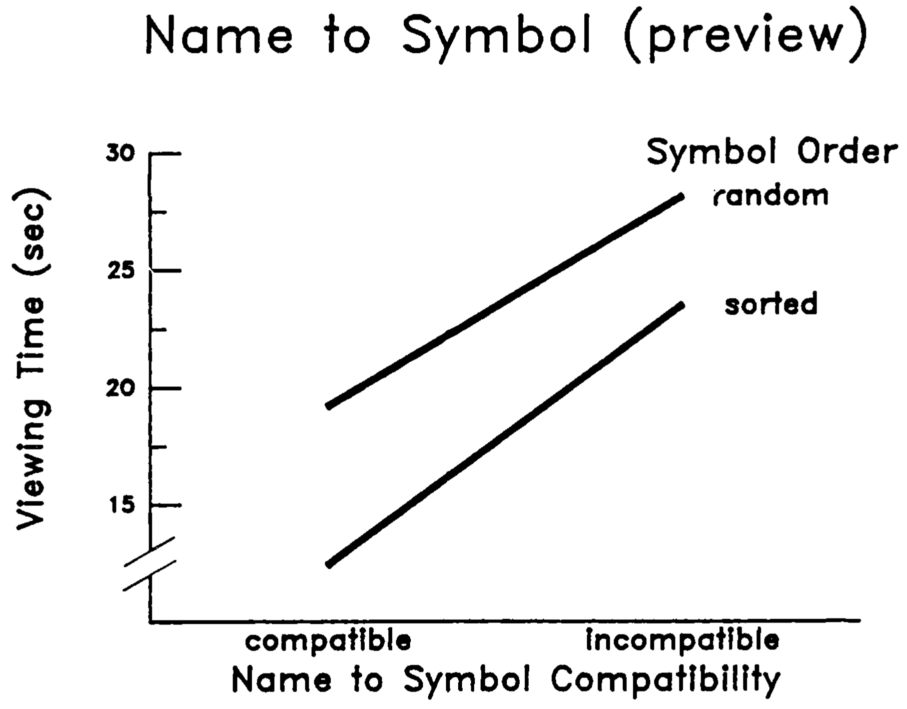
There was an advantage of compatible symbols over incompatible ($\underline{F}(1,14) = 107.9$, $\underline{p} < .001$), but the effect was very different for letter symbols than words ($\underline{F}(1,14) = 52.1$, $\underline{p} < .001$). Compatible letters (1139 msec) were the easiest symbols, while incompatible letters were the most difficult (2224 msec). Compatible numbers were superior to incompatible numbers, but the advantage was not as great as that for letters (454 msec compared to 1085 msec). A 99 percent Scheffe confidence interval showed all pairs of means to be significantly different. Sorted words were superior to randomly ordered ones ($\underline{F}(1,14) = 6.9$, $\underline{p} < .02$), 1658 msec compared to 1726, an advantage of 68 msec. No other experimental manipulations showed effects, nor were their any additional interactions.

Figure 6: Tradeoff Between Compatible and Incompatible Symbols



Shown here are the main results of Experiment 2 in which subjects had to generate symbols for names. Compatible letter symbols were the easiest (significantly faster than compatible numbers), while incompatible letters were the worst (significantly worse than incompatible numbers). The advantage for compatible numbers over incompatible numbers is also reliable.

Figure 7: Time Spent Previewing Name to Symbol Lists



Shown here are the mean times subjects spent studying the lists they were to learn. Subjects spent significantly more time studying lists with randomly ordered symbols than sorted, and significantly more time studying lists with incompatible name to symbol pairings. There were no differences for lists of different symbol types.

Previewing Times

Previewing times for lists are shown in Figure 7. They can be used as an index of the perceived complexity of the displays for subjects. Subjects spent less time previewing compatible displays than incompatible ($F(1,15) = 14.9$, $p < .01$), and less time studying displays with sorted symbols than random ($F(1,15) = 8.5$, $p < .01$). The order of the target words had no significant effect on preview time ($F < 1$), nor was there an effect of whether selectors were letters or numbers ($F < 1$).

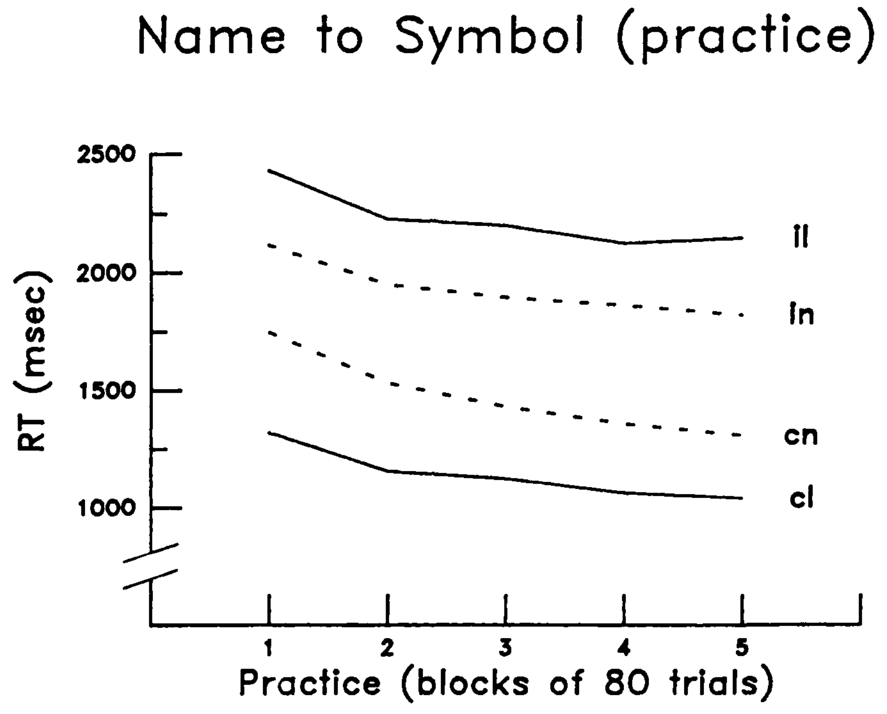
Practice Effects

There was a reliable effect of practice over the five blocks of trials within each condition ($F(4,56) = 35.0$, $p < .001$), but there was no interaction with any other factor. Evidently, the effects of symbol type and compatibility are persistent, and this is shown in Figure 8. Compatible letters (cl) are consistently the best symbols, followed by compatible numbers (cn) followed by incompatible numbers (in) followed by incompatible letters (il), and the relative size of the effects are stable over practice.

Errors

Error rates for all conditions were very low, averaging less than one percent of trials. Most errors can be attributed to hitting a key next to the key that should have been hit (e.g., typing "s" instead of "a" or "v" instead of "b"). After such uninteresting errors were removed from the error analysis, there were only 16 errors left, too few for significance tests, though most of the errors were in the incompatible conditions. Errors were evenly split between number and letter conditions.

Figure 8: Stability of Name to Symbol Compatibility



Shown here are the mean times to type the symbol paired with a name broken down over compatibility and symbol type conditions, showing the consistent differences even with practice. Compatible letter symbols (cl) are fastest, followed by compatible numbers (cn), incompatible numbers (in), finally followed by incompatible letter symbols (il).

Discussion

The results from this experiment are clear, and similar in nature to those of Experiment 1. Given a name, it is trivial to generate the symbol for it if the symbol is the name's first letter. But if the symbol is some randomly chosen letter, the activation of the first letter of the name competes with the symbol, and this results in huge differences in response time: over a factor of two. This is impressive compared to the response times obtained with number symbols which yielded intermediate results.

General Discussion of Symbol <==> Name Experiments

Experiments 1 and 2 are consistent with the extension of McClelland and Rumelhart's (1981) model of word perception applied to associating symbols with names in artificial languages. The data presented here are consistent with the view that different symbols have different associative strengths with a name. The first letter of a name is a better symbol for it, and this superiority does not seem to be due to an initial advantage of memory retrieval. First, there is evidence that there are inhibitory associations between incompatible letters and names because incompatible letters were consistently inferior to the more neutral number symbols (both compatible or incompatible). Second, the advantage for compatible letters over incompatible letters actually increased with practice. If it were some simple memory strategy at work instead of a basic linguistic process, the effects should go away over time. Though people got faster at the tasks with practice, the differences in response times between different symbol-name pairings were resilient to change.

The error analyses, though largely inconclusive because of the low error rates, suggest that activations between symbols and names compete to produce what Norman (1980) called activation errors, a generalization of the Stroop (1935) phenomenon. Stroop (1935) presented colored words and had people name the colors of the words. The words were names of colors, sometimes the same as the color of the word ("red" was printed in red), sometimes different ("blue" was printed in orange). When asked to name the colors, the people were slower and made more errors when the name was incompatible with the color to be named. Norman's theory of competing activations is supported more by the evidence that the competing activations cause slowdowns in response times of about a factor of two. In some situations, where response speed is stressed more than accuracy, I would expect to see more of the slow responses associated with incompatible symbols being replaced by errors.

Combined, Experiments 1 and 2 show the tradeoffs involved in choosing symbols for names. Using letter symbols can be the most efficient scheme, but only under certain circumstances. When incompatibilities might arise, more neutral symbols are preferable, showing that the more one knows about the domain of discourse for an artificial language, the better one can do in terms of optimizing communication.

NAMES <==> CONCEPTS

Experiments 1 and 2 dealt with associating symbols with names. The results showed that when a person has to associate a symbol with a name, either by recalling the name given the symbol (comprehension of notation) or by generating the symbol given the name, the compatibility between symbols and names have reliable effects on the time it takes to make the association. These effects can be facilitory or inhibitory compared to neutral symbol-name pairings, as shown by comparisons with the control: number symbol conditions. In this section, I further investigate the interactive model of processing linguistically mediated artificial language, but now studying the associability of names with concepts.

I want to show that the name chosen for a concept affects how that concept is interpreted. I do this by a priming technique that lets me observe the effects of choice of names on symbols around the names. This is based on my earlier discussion of the suggestiveness of names where I claimed that the name for a concept serves as a key to memory structures in terms of which it, and other concepts, are interpreted. When reading expressions in artificial language, people build representations of their meanings as they are read. Using names specific to a domain is more concrete than using general names, and so gives people more hints about the relationships of the terms, though, as Halasz and Moran (1982) point out, some of these hints might be misleading if the terms are poorly chosen. General terms carry less information, and so should be less useful where specific terms would be appropriate. General terms would be more useful when the domain of application is less clear, and generalization to many domains is a desirable property.

A Motivating Example: Names <==> Concepts

Experiment 3 is motivated by practical experiences, as are the previously reported experiments. The experimental question grew out of discussions about how to choose a name for a program: whether a program should be given a general name or a specific one. For example, on our computing system, we have several databases to store information. One is an on-line list of publication references used to help write papers. People who want to include references in their papers specify the authors of the publications, and their dates, and a program called REF is used to gather the full references with book names, journal volumes, etc., filled in. Another database keeps records on the addresses of people, and records from it are found by specifying a name to a program called GRAB. These two databases are searched by nearly identical programs; the records holding the information are in roughly the same format. Still, many people who know about the REF program do not know about the GRAB program, and vice versa. Some people who know about both do not realize they are working with about the same program applied to difference databases, though this is partly because the programs are slightly different. The question is whether people would be better off with one program, perhaps called SEARCH, that could be used on both of these and other databases.

Consider how the model of processing linguistically mediated artificial languages would predict the outcome of an experiment testing this question. I depart from the actual form of the commands used to call REF and GRAB for the sake of exposition. Suppose you can specify the author(s) or date of a publication to REF and the name or occupation of a person for GRAB. A call of the REF program might look like:

```
REF a=Hofstadter d=1980
```

which would look for publications by Hofstadter from 1980. A call to the GRAB program might look like:

```
GRAB n=Smith o=psychologist
```

which would look for records on psychologists named Smith. In these examples, an argument to either program is a letter specifying the property to "key" on, followed by the attribute to search for under the specified property. A person reading these program calls goes through a process like that described in the earlier description of the DESC program. The name of the program begins activating concepts related to it, and these in turn activate the names of concepts associated with them. For example, REF activates "references" which activates concepts like "author," "subject," "title," "date," and so on. These activated names in turn activate letters, notably their initial letters: "author" activates "a," and "date" activates "d." A similar process, but with different concepts, names, and symbols, occurs with GRAB, though it is not clear that GRAB is a very suggestive name for "person" properties.

The model predicts that a suggestive name of a program can facilitate the processing of the symbols around it. A name that does not suggest what it can be used for--GRAB is a good example--will not facilitate the processing of symbols around it, but a relatively neutral name may be better for general use, because it will not bias people toward one specific use. If the suggestively named REF program were used on the database of addresses, as well it could be, would people have problems constructing and comprehending the meanings of commands? The model predicts inhibitory associations as well as facilitory, and so predicts that problems would arise. When a program is thought to be useful for one application because of its suggestive name, it produces a cognitive set against using it in novel situations. Even if this set is overcome, Experiment 3 shows that what is a suggestive name for one application can be a confusing name for another, and that for programs expected to have many applications, more general, less specific or suggestive names are in order. Applying these concepts to artificial languages in general, the name chosen for a concept affects how concepts around it are processed, and that names for concepts can be chosen to be maximally suggestive only at the cost of being minimally general.

Bernard et al (1982) studied the issue of general versus specific command names and found that subjects using general command names for a text editor mockup more readily and more often requested help with the commands. Evidently, people understood that the general names provided less information and compensated for it, however, no differences were found between general and specific names in terms of the accuracy with which subjects recalled or recognized them. Rosenberg (1982) discusses the suggestiveness of command names claiming that a command name is "good" to the extent that the name directly suggests what the command does, and to the degree that it directly suggests the relationships (e.g., similar, opposite, unrelated) of that command to others in the system. Rosenberg provides a method of evaluating the suggestiveness of names by measuring their judged similarity to examples of what the commands do, however, his method does not bear on how the names are processed by people using artificial languages such as the text editor language he studied. As far as the literature on naming computer commands is concerned, the use of memory tests instead of process timing tests, makes their application to a processing model of artificial language impossible. Black and Moran (1982) point out the need to go beyond such global effects and study the underlying psychological factors involved in naming commands for programs, and I might add, artificial languages in general.

To demonstrate the tradeoffs of specificity versus generality of names, I use a priming technique based on the symbol-name learning tasks of Experiments 1 and 2. I have people learn names for symbols and demonstrate their learning by having them speak the name paired with the symbol, much as in Experiment 1, but in the next experiment, I present a priming context before the symbol is presented. In Experiment 3, the names paired with symbols are properties of objects (e.g., age, height, and weight are some for a person, and climate, government, and population are some for a country). Each list of name-symbol pairs to be learned was made from two sets of five properties of two different objects (e.g., a person and a country). The priming context for a symbol is an appropriate suggestive name, a neutral name, or an inappropriate suggestive name for the purpose of the symbol.

Exper'ment 3: Names ==> Concepts

In this experiment, I investigate the effect that the name for a concept has on how people interpret that concept and concepts around it. I manipulate the specificity or suggestiveness of names for concepts. I test specific names for concepts, that provide a lot of information about their purposes, against general names for concepts that tell relatively little. The general paradigm is to have subjects learn a list of names of properties so they are able to recall the names when presented with the first letter of the names. This is the compatible letter symbol-name task of Experiment 1 which was the easiest for subjects to perform. The properties in a list are divided into two subsets, one of properties of one object, such as a person, one for another, such as a country. Before a letter is presented, a priming context is presented: either a semantic context indicating the object to which the upcoming property applies, or an alphabetic context, indicating which half of the alphabet the letter will be in. The letter symbol for a name appears and subjects have to say the name paired with the symbol, the word in the list beginning with the letter symbol.

Method

Materials

Two sets of stimuli were constructed of five properties of four objects: a person, a country, an animal, and a book. Person properties were paired with country properties, and animal properties were paired with book properties. The lists were:

PERSON COUNTRY	BOOK ANIMAL
birthday	author
climate	breed
government	editor
height	habitat
industry	intelligence
land	mobility
occupation	nutrition
population	publisher
religion	subject
weight	title

There are five properties for each object, making two lists of ten properties. Each list has five properties lexicographically preceding words beginning with the letter "k," and half following. A different random permutation of the lists was used for each subject, so that the cue of being before or after "k" and the cue of being a property of one or another type was uncorrelated over subjects with serial list position.

Procedure

At the beginning of the experiment, the list a subject was going to learn was displayed in the upper left corner of the screen of a dedicated micro-computer. This list remained present for subjects to see for the whole experiment. Half the subjects learned the PERSON-COUNTRY list while half learned the ANIMAL-BOOK list. On each trial, a priming context described in the next paragraph appeared in the lower right quadrant of the screen, followed by a pause of random duration between one and two seconds, followed by the presentation of the first letter of one of the words from the list of ten properties. The task was for subjects to speak the name of the word beginning with the presented letter as quickly and accurately as possible. Their responses were timed by a voice-key relay connected to the computer, and recorded on a tape recorder for error analyses.

There were four conditions of interest. Two _types_ of primes were used: semantic primes told the names of the objects to which the upcoming stimuli applied (e.g., PERSON or COUNTRY), and alphabetic primes told whether the upcoming stimulus was before or after "k" in the alphabet. The primes provided roughly independent and equal amounts of information about which five stimuli were going to appear, one based on the semantics of the stimulus, the other based on alphabetic information. Another factor was whether the priming information was _appropriate_ for the upcoming stimulus. Eighty percent of the time, the priming context was appropriate for the upcoming stimulus. For example, The prime was "PERSON" and the stimulus was "w" for which the subject would say "weight." Twenty percent of the trials had inappropriate primes. For example, in the inappropriate alphabetic condition, they were told that the upcoming stimulus would be _greater_ than "k" but the stimulus was "b" (which is _less_ than "k") for which they would say "birthday."

The alphabetic prime served as a control condition for the semantic prime in two ways. First, it was a non-semantic prime, so it presumably did not prime concepts (i.e. the properties) related to either of the two objects whose properties made up the list; it was not expected to make a cognitive set. Second, it provided the same amount of information as a semantic cue, because it restricted the set of probable stimuli to one half the list, as did semantic cues. If it was used by subjects to predict which half of the items would appear, that information was independent of the semantic division of the list.

Subjects went through twenty random permutations of the ten elements of the list. The order of the experimental conditions, 80 percent appropriate, 20 percent inappropriate, and half semantic primes, half alphabetic, was a different random permutation for each of the ten item lists for each subject. The order of property names in the list was random for each subject so that neither semantic nor alphabetic cues could help subjects' list searching strategies. The 200 trials were presented with two second pauses between and took subjects about twenty minutes to complete the experiment.

Subjects

Subjects were 16 paid students and staff of the University of California San Diego.

Results

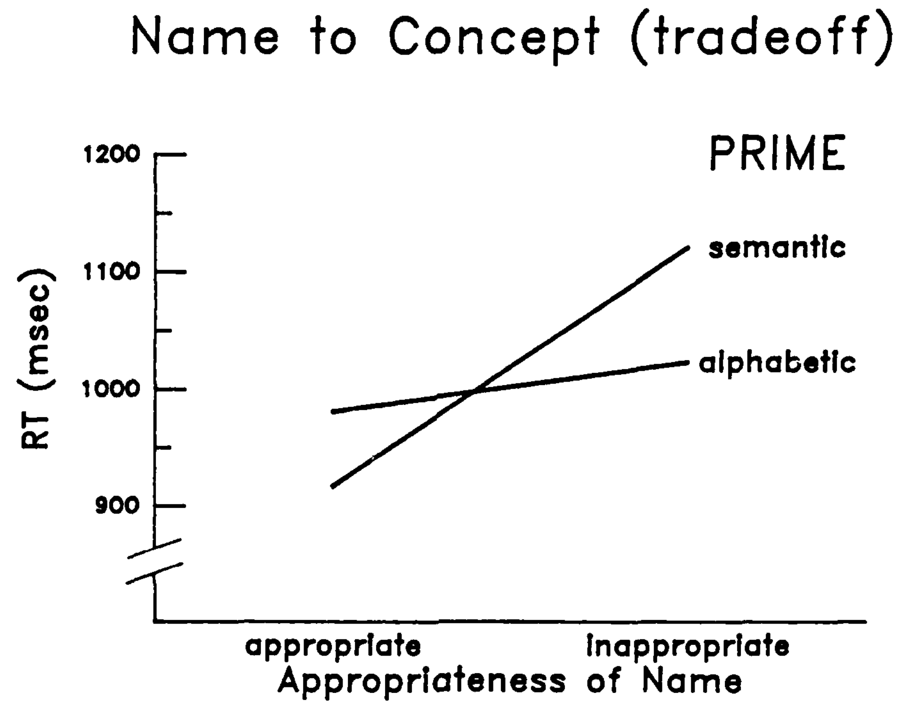
Major Results

Figure 9 shows the interaction of appropriateness and type of prime ($\underline{F}(1,14) = 8.0$, $\underline{p} < .02$). The mean reaction times for speaking the name paired beginning with the stimulus letter are shown below:

Appropriateness	Prime	Mean RT
appropriate	semantic	917
	alphabetic	981
inappropriate	semantic	1121
	alphabetic	1023

A Scheffe 95% confidence interval based on the log transformed response times showed any differences greater than 60 msec to be significant, thus all the conditions were reliably different except for the two alphabetic prime conditions which differed by 41 msec. The data show that semantic primes help subjects interpret symbols appropriate with the primes (compared to the control condition of appropriate alphabetic primes), but have inhibitory effects (compared to the inappropriate

Figure 9: Tradeoff Between Specific and General Names



Shown here are the main results of Experiment 3. Appropriate semantic priming contexts (e.g., BOOK-t-title) produced a facilitory effect (compared to appropriate alphabetic primes) on subjects' ability to say the name beginning with the stimulus letter. Inappropriate semantic primes (e.g., ANIMAL-s-subject) had the opposite effect: subjects were significantly slower compared to the control, inappropriate alphabetic primes. The two alphabetic priming contexts do not differ significantly. Appropriate semantic primes correspond to suggestive names used in the specific context for which they are appropriate, while inappropriate semantic primes correspond to those names used in an inappropriate context. Alphabetic primes serve as a control, being equally discriminating for list elements, and corresponding to a neutral, or general name.

alphabetic prime control) when the prime is inappropriate.

There was an overall advantage for appropriateness of prime ($\underline{F}(1,14) = 9.9$, $\underline{p} < .01$), primarily due to the advantage of appropriate semantic over inappropriate semantic primes.

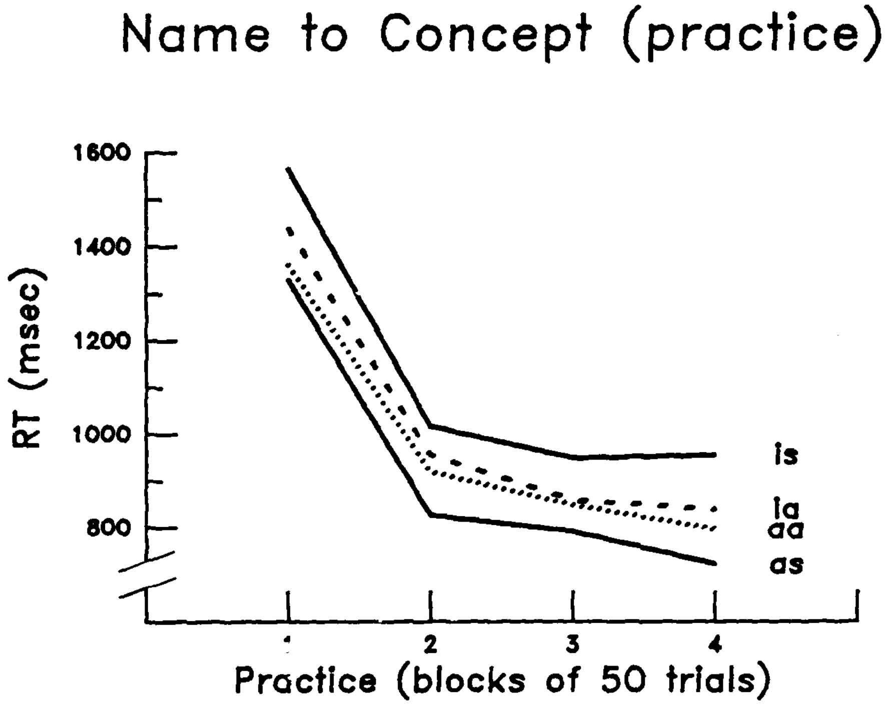
Practice Effects

The effects shown in Figure 9 are stable over practice. Of course, subjects got faster as they went through the experiment ($\underline{F}(3,42) = 49.7$, $\underline{p} < .001$), and this is shown in Figure 10. A Scheffe 95% confidence interval showed that subjects got significantly faster during the first three blocks of 50 trials of practice, but that the last block did not significantly differ from the next to last. Another Scheffe 95% confidence interval showed that appropriate semantic primes resulted in superior performance to inappropriate throughout the experiment. The effects relative to the control alphabetic primes were significant about half the time, evenly distributed over practice. Figure 10 shows the effects of practice on response time, broken down over prime type and appropriateness conditions. The effects of prime and appropriateness are remarkably stable over time, and this suggests that the effects of appropriate of name do not go away with limited practice.

Errors

The overall error rate was about 4.5 percent, or 127 errors in all. Most errors were repetitions of responses not loud enough for the voice-key apparatus to detect. These 48 soft responses resulted in inordinately long reaction times and were excluded from the analyses. The occurrence of such uninteresting errors were uncorrelated with any experimental conditions. Almost all the remaining errors could be classified as activation errors of the type observed in Experiment 1. For example, if the stimulus was "c" and the correct response "climate," a property of "country," an activation error would be if the subject said "country." No class of errors showed any significant correlation with any experimental condition, including practice (subjects did not

Figure 10: Stability of Name to Concept Tradeoff



Shown here are the means of the four main conditions of Experiment 3, broken down over practice. Through each block of fifty trials, the order of the conditions is the same, showing the persistence of the priming effects over time. From fastest to slowest: appropriate semantic (as), appropriate alphabetic (aa), inappropriate alphabetic (ia), inappropriate semantic (is). The only reliable differences over all blocks of practice are those between appropriate and inappropriate semantic primes.

get more or less accurate with practice, only faster).

Discussion

The semantic primes had facilitory and inhibitory effects on how easily symbols around them were associated with the names they represented. The appropriate semantic primes suggest what symbols are likely to appear around them and so are comparable to choosing a specific, suggestive name for a concept. The alphabetic primes, which theoretically provided the same amount of anticipatory information as the semantic primes by restricting the number of elements in the lists that were likely to appear, was not as useful to subjects, perhaps because their semantic knowledge of the stimuli was more useful than their lexical knowledge, which would in general be less useful. For that reason, the alphabetic primes are comparable to choosing a general name for a concept, because they do not have the same activation effect. Finally, the inappropriate semantic primes are comparable to the case where a specific name is chosen for an advantage in one application domain, but the name is then used in another domain, causing activation of concepts in an inappropriate domain, and inhibition of the concepts for the new domain.

The results of Experiment 3 are consistent with those found in semantic priming experiments. Facilitory effects of semantic priming have been observed in experiments described by Collins and Loftus (1975), and both facilitory and inhibitory effects have been observed by Neely (1977). Neely found that appropriate semantic primes (e.g., BIRD) helped subjects decide that a semantically related stimulus (e.g., robin) was a word, compared to a control condition, presenting a string of X's, that provided no priming information. Neely also found that inappropriate semantic primes (e.g., BIRD) hindered subjects decisions that a semantically unrelated stimulus (e.g., arm) was a word, compared to the same control condition. There are two differences between Neely's experiment and Experiment 3. (1) Neely's semantic priming effects were between whole-part (e.g., BODY-arm) or super-subordinate (e.g., BIRD-robin) relationships while Experiment 3 tested object-property relationships. More importantly, (2) In Experiment 3, I wanted to test if the semantic priming effects carried through to the process of associating a symbol with its paired name. This second difference was important to find support for the interactive model of processing linguistically mediated artificial language which posits multi-level interactions between symbols, names, and concepts.

The implications of this experiment are that a domain specific name for a concept provides an advantage for that domain, but causes a disadvantage for application to other domains. If the name is to be used in only one context, where it is guaranteed to be appropriate, then there is not a problem. Using a specific name in one novel context would result in one appropriate use and one inappropriate, which would be about the same as having a general name applied to both. However, if the name may be used in many contexts, then a general name would give the best overall results.

PRACTICAL APPLICATIONS

In this section, I show how the results of Experiments 1, 2, and 3, along with general results from cognitive psychology, can be applied to the design of natural artificial languages. I think demonstrating their application is important for several reasons.

(1) An important test of any theory is whether its predictions are sound. Therefore, application is an informal test of my ideas about artificial language processing.

(2) It is useful to apply psychology to the design of anything people will use because it can make it better: easier to use, less prone to cause errors, etc.

(3) It is no small secret that the support supplied to research depends strongly on its applicability.

For those reasons, I discuss the applicability of research on artificial languages and cognitive psychology in general to the design of a user interface to a programming system. The user interface is called MENUNIX (Perlman, 1981). It is a menu-based interface to the files and programs of the UNIX (Richie & Thompson, 1974, 1978) operating system, developed and trademarked by Bell Telephone Laboratories. Though many stages of the design of MENUNIX were guided by the application of psychological principles, I describe only those parts of MENUNIX whose design was directly affected by the work on artificial languages presented here.

MENUNIX: A Menu-Based Interface to a Programming System

MENUNIX is a menu interface to UNIX files and programs. It always displays two menus from which users can make selections, each at a fixed location on the terminal screen. Figure 11 shows the main parts of the MENUNIX display. The FILE MENU displays the files in the current working directory in a window on the right side of the screen. The PROGRAM MENU displays the programs of UNIX in a window on the left. Both the FILE MENU and the PROGRAM MENU are hierarchically structured, the FILE MENU following the UNIX file structure, and the PROGRAM MENU organized so that programs with similar purposes are grouped together into units I call workbenches. Both FILE and PROGRAM MENU entries can be selected with single keypresses, displayed on the left of each entry. Programs and workbenches are selected with mnemonic letters, while files and directories are selected with numbers.

The FILE MENU

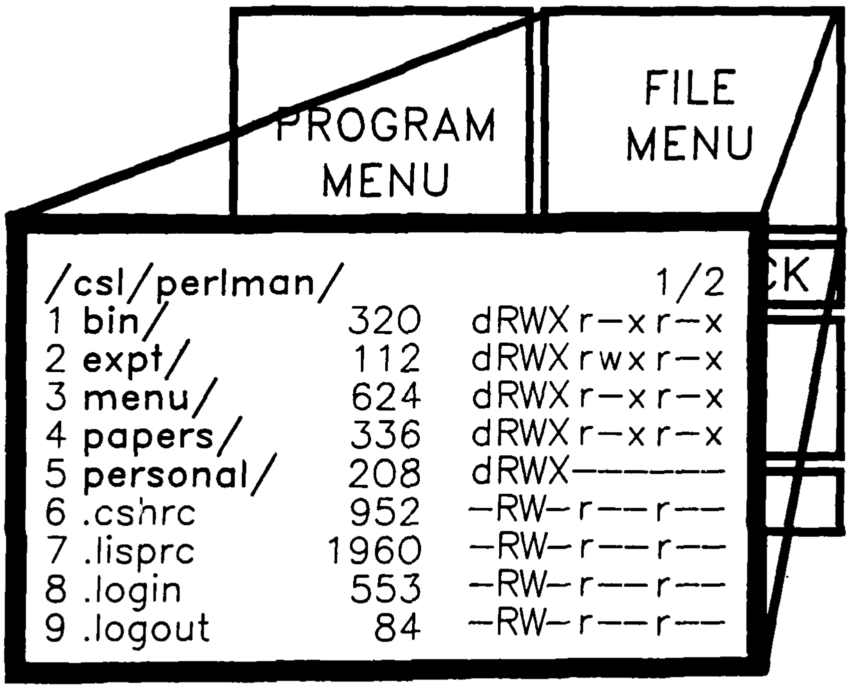
On the right side of the MENUNIX display is the FILE MENU that displays the UNIX file system. A detailed figure of the MENUNIX FILE MENU is shown in Figure 12. At its top is the full pathname of the current working directory, followed by the page number and number of pages of that directory (3/5 means you are on page three of a total of five). Directories typically have more entries than can be displayed on a screen, so each directory is divided into pages, each containing a manageable sized part. At most nine files are displayed at one time, and these are selected with the numbers 1-9 displayed next to file names. Users can leaf through the pages of a directory by using the plus and minus keys. Typing a plus goes to the next page (if there is one), and typing a minus goes to the previous one.

Figure 11: MENUNIX PROGRAM and FILE MENUS

```
[Writing aids]                               /csl/perlman/papers/artlang/thesis/  1/2
a    Analyze style (style)                   1   chap1           41712 -RW-r--r--
c    Count lines, words, and chars (wc)      2   [illegible]     50853 -RW-r--r--
d    Decode/Encode (crypt)                   3   chap3           19916 -RW-r--r--
e    Edit a file ($editor)                   4   chap4           14589 -RW-r--r--
f    [Format text file]                      5   figures          8063 -RW-r--r--
h    Heading structure (headings)            6   macs             1667 -RW-r--r--
l    Look for word in dictionary (look)      7   print              30 -RWXr-xr-x
p    Permuted indexer (ptx)                  8   pubs            81567 -RW-r--r--
r    Reference finder (pub)                  9   refs             9047 -RW-r--r--
s    Spelling corrector (correct)
t    Typo finder (typo)
w    Wordy sentence finder (diction)
```

Shown here are the two main parts of the MENUNIX display. On the left half of the display is the PROGRAM MENU which displays programs and workbenches, which are collections of programs. Entries in the PROGRAM MENU are selected by pressing the mnemonic letters displayed to the left of the phrase describing the entries' names. On the right half of the display is the FILE MENU which displays files and directories, which are collections of files. Entries in the FILE MENU are selected by pressing the numbers displayed to the left of the files' names.

Figure 12: MENUNIX FILE MENU



```
/csl/perlman/                   1/2
1 bin/          320  dRWX r-x r-x
2 expt/         112  dRWX rwx r-x
3 menu/         624  dRWX r-x r-x
4 papers/       336  dRWX r-x r-x
5 personal/     208  dRWX------
6 .cshrc        952  -RW- r-- r--
7 .lisprc      1960  -RW- r-- r--
8 .login        553  -RW- r-- r--
9 .logout        84  -RW- r-- r--
```

Shown here is a close-up of the MENUNIX FILE MENU. The top of the FILE MENU displays the current working directory, and its page number/number of pages. Each entry includes a numerical selector, the name of a file, the size of the file, and its UNIX access protection code. Directories are displayed at the beginning of the display and are highlighted by being in bold font.

To select a file, users type the number on the left of its name. The effect of selecting a file depends on the type of file selected, which can be determined from the information displayed on the right of its name. To the right of a file name is displayed its size (in characters), and its access protection modes. The access protection modes contain ten characters interpreted as follows. The first character indicates the type of file, usually a plain file (-) or a directory of files (d). The next nine indicate independent read, write, and execute permissions for the owner of the file, the owner's group, and all other users, respectively.

When a FILE MENU entry is selected by pressing the number key on its left, MENUNIX tries to do the sensible action based on the type of entry selected. Selecting a directory causes the working directory to change to that directory. Directories are identified by their access modes, by being followed by a slash, and on some terminals by being highlighted. Selecting an executable file (a program) causes that program to be run after arguments are requested. Selecting a plain file calls the user's preferred editor on that file.

The number key 0 is not used to select a file but is reserved for going "upwards" in the file system. While pressing the number key to the left of a directory name goes into the selected directory, the 0 key changes the working directory to the parent directory in the UNIX file system hierarchy. In summary, in MENUNIX, the contents of directories are always present for the user. Users are able to edit files and change directories with just the FILE MENU commands.

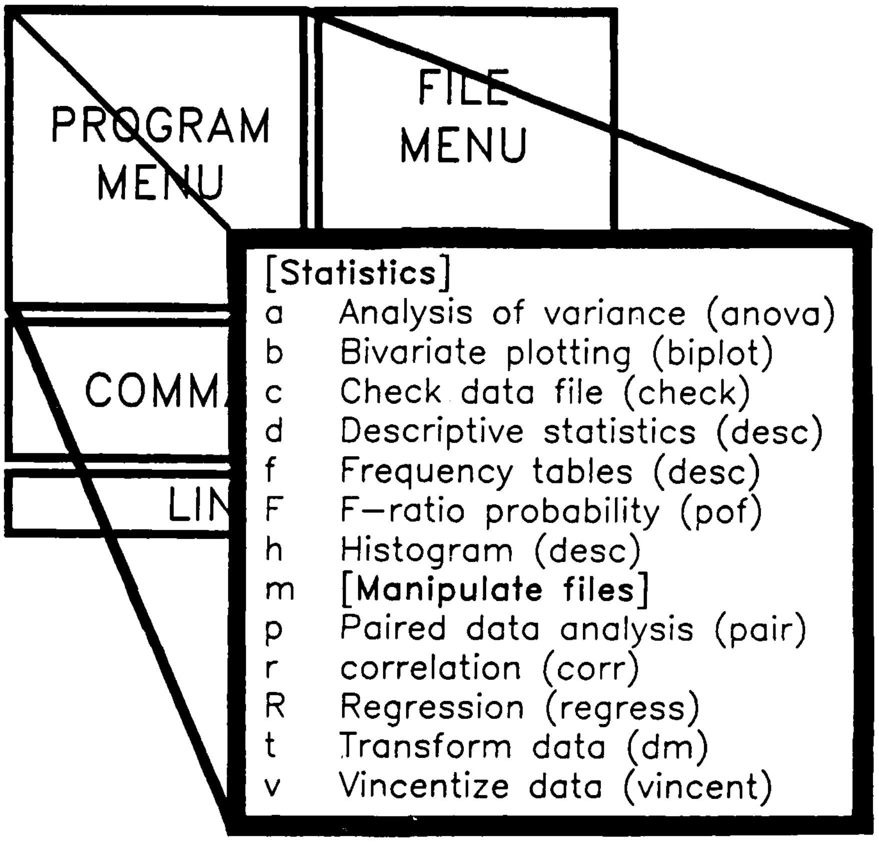
The PROGRAM MENU

Though UNIX implementors suggest organizing files by making content-oriented directories, one for programming, one for writing, etc., UNIX programs are not clustered by function, but by the installation where they were written. Programs are kept in special directories based on whether they were developed at Bell Laboratories, The University of California at Berkeley which distributes an enhanced version of UNIX, or whether they were developed at a local installation. Partly because of this, it is difficult to find programs of interest from the hundreds that exist.

On the top left of the MENUNIX display is the PROGRAM MENU, a closeup of which is shown in Figure 13. The PROGRAM MENU organizes programs into a hierarchy much as the file system can be used to organize files. Entries in the PROGRAM MENU are either programs, or collections of programs, called workbenches. This is analogous to entries in the FILE MENU being files, or collections of files, called directories. At the top of the PROGRAM MENU is the name of the workbench in use. The initial workbench is [UNIX] (workbenches are displayed in [square] brackets). [UNIX] contains the most general classification of UNIX commands, so general that it does not contain any programs, but only names of task specific workbenches that contain all and only those commands that are of interest to people working on a specific task. For example, there is a programming workbench that contains more specific workbenches for general programming and specific programming languages: C, FORTRAN, LISP, Pascal, etc. There are other workbenches for writing papers, sending and receiving mail, and so on.

Each line of the PROGRAM MENU consists of a mnemonic letter used to select an entry (program or workbench), followed by a short phrase describing the entry. Menu selection in the PROGRAM MENU has results analogous to those in the FILE MENU. Selecting a workbench causes the PROGRAM MENU to change to and display the new workbench. Commands are also available to get back to the root of the PROGRAM MENU hierarchy.

Figure 13: MENUNIX PROGRAM MENU



[Statistics]

a	Analysis of variance (anova)
b	Bivariate plotting (biplot)
c	Check data file (check)
d	Descriptive statistics (desc)
f	Frequency tables (desc)
F	F-ratio probability (pof)
h	Histogram (desc)
m	**[Manipulate files]**
p	Paired data analysis (pair)
r	correlation (corr)
R	Regression (regress)
t	Transform data (dm)
v	Vincentize data (vincent)

Shown here is a close-up of the MENUNIX PROGRAM MENU. The top line of the menu displays the name of the current workbench, in this case, a workbench for doing statistics. Each line of a workbench displays the mnemonic selector for the entry followed by a short descriptive phrase. Workbenches are bracketed and displayed in bold font to make them more visible. Programs are followed by their true UNIX names in parentheses (e.g., desc).

Program entries are displayed followed by the parenthesized name of the UNIX program they call. This is useful so novices can learn the real names of programs and for experts so they know what familiar programs they are selecting. Naturally, selecting a program causes it to run.

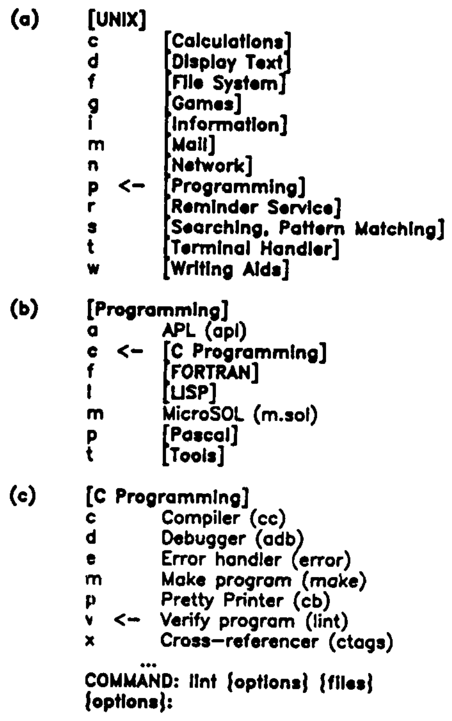
An example of a sequence of selections leading to the C program verifier, lint, is shown in Figure 14. It is called lint because its writer thought the name cute; the program picks up pieces of loose fluff from programs. The figure shows one of the virtues of a content hierarchy. The poorly chosen name for the C program verifier does not hinder a user's ability to find the program.

Application 1: Symbols <==> Names

With this brief introduction to MENUNIX, we can now see how the general psychological theory and specific experimental results were applied to the design of this user interface. Based on the results of Experiments 1 and 2, we can conclude there is no system of choosing symbols that is most efficient for all situations. If the pairing between names and symbols can be controlled so that activations do not compete, using letter symbols is optimal. But if such control can not be obtained, competing activations between symbols and first letters of names will result in much worse performance.

An early decision in MENUNIX's design was to use single character selectors for entries in both the PROGRAM and FILE MENUS. This was to make menu selection faster. The results from Experiment 2 indicated how to select them, because choosing selectors for menu entries is the same thing as choosing symbols for names. The structure of the two menus demanded different menu selection schemes. First note that the display of programs is not likely to change from day to day. It is a static menu. The menu of files is one that we would expect to change very often, probably every day. It is a dynamic menu. These two types of displays require different option selection schemes: mnemonic letters for the static display of programs and ordered numbers for the

Figure 14: A MENUNIX Program Selection Sequence



```
(a)     [UNIX]
        c         [Calculations]
        d         [Display Text]
        f         [File System]
        g         [Games]
        i         [Information]
        m         [Mail]
        n         [Network]
        p   <-    [Programming]
        r         [Reminder Service]
        s         [Searching, Pattern Matching]
        t         [Terminal Handler]
        w         [Writing Aids]

(b)     [Programming]
        a         APL (apl)
        c   <-    [C Programming]
        f         [FORTRAN]
        l         [LISP]
        m         MicroSOL (m.sol)
        p         [Pascal]
        t         [Tools]

(c)     [C Programming]
        c         Compiler (cc)
        d         Debugger (adb)
        e         Error handler (error)
        m         Make program (make)
        p         Pretty Printer (cb)
        v   <-    Verify program (lint)
        x         Cross-referencer (ctags)
            ...
        COMMAND: lint {options} {files}
        {options}:
```

Shown here are close-ups of successive displays of the PROGRAM MENU the sequence of workbenches displayed while finding the C program verifier (lint). Beginning in the [UNIX] workbench (a), typing "p" causes the [Programming] workbench to be displayed (b). In this context, typing "c" changes the display to the [C Programming] workbench in which the C program verifier is found (c). Selecting the verifier causes the MENUNIX line-editor (not described, but for users to input information) to request options and files to be verified.

dynamic display of files. The reasons for this design decision are explained below.

One problem with using mnemonic initial letter selectors is the possibility of two menu options having the same first letter. The MENUNIX user interface was developed for a system with hundreds of programs and thousands of files. Clearly, there can not be a unique symbol for each entry in the program or file menu. MENUNIX solves this problem by hierarchically structuring the programs by content so that any particular menu of programs offers about ten programs, a cognitively manageable number (Miller, 1956). Mnemonic letters can not be used for selecting files from a menu. With a small static set of programs for each menu, it is possible to rename programs using synonyms to insure unique first characters, a practically unworkable scheme for files. It would be difficult to control the names of files because of the frequency with which files change. For example, in a menu of programs for writing papers, there might be a program to correct spelling and one for style analysis. On the system for which MENUNIX was developed, the programs are called <u>spell</u> and <u>style</u> respectively. Alternative names that would solve the problem of clashing initial letters are <u>correct</u> and <u>analyze</u>. Such a renaming scheme for files would be too intrusive for users.

One possibility for a selection scheme for files in a menu is a list of sorted letters. This scheme would not work out well because of possible competing activations between the selecting symbol and the name of the file because the pairing would tend to be random, producing close to the same conditions as incompatible letter symbols in Experiment 2. Suppose there is a menu of files, and the fourth file in the list is called "paper."

a	data
b	letter
c	notes
d	paper
e	references

With sorted letter selectors, the character used to select "paper" would be "d." The first character of "paper," "p," would compete with the selector "d" and the result would be slower selection times, and perhaps errors in selection. Similarly, having to select the file called "data" with "a" would result in competing activations, this time between "d" and "a," possibly resulting in selecting "paper" by mistake because it is selected with the first letter of "data." Because of these problems, the decision was made to use the more neutral sorted number selectors that yielded performance only slightly worse than the mnemonic first-letter selectors that could not be used.

Choosing separate selection schemes for programs and files had the added attractiveness of having disjoint sets of selectors for programs and files. This made it unnecessary to specify from which menu the selections were being made, removing the need for different selection modes: one for selecting programs, one for files. This removed the possibility that a person would think a keypress applied to one menu when the other was active, an error Norman (1981) calls a mode error.

Application 2: Names <==> Concepts

In Experiment 3, I discuss how the name for a concept affects how people interpret concepts around it, a measure of how the concept itself is being interpreted. I now turn to a discussion of how to choose names for concepts so that when a person has a concept in mind, the name for it can easily be accessed. My goal is to show how the results of Experiments 1, 2, and 3 can be applied along with previous psychological research, to the design of notational systems that accommodate to strengths and weaknesses of human cognitive abilities.

Black and Moran (1982) studied the learning and remembering of command names and found that specific terms were superior to general terms for learning and recall and that infrequent names (as measured by the Kucera and Francis (1967) word frequency norms) were superior to frequent. Baggett (1980) outlines a method for naming objects with an iterative procedure that maximizes the ability of people to recall the names for them. Despite attempts to find rules for naming commands, severe problems exist for people trying to learn large numbers of them. While research on naming small sets of commands might provide us with some insights about how to choose names in general, it is little help for very large sets. On the computing system we use in our laboratory, there are several hundred commands, each with its own peculiar name, often an abbreviation or amusing name that provides people with little or no idea of its purpose. People generally learn about commands when they are told about their existence by an expert. Consulting a manual is little help; though the manual is alphabetized, the names are not suggestive of their purposes, and reading each manual entry would be prohibitively time consuming.

A partial solution to this problem developed from experiences with MENUNIX. This solution resulted in the invention of a novel naming system for programs based on how they can be arranged in a hierarchy, the means of organizing information Mandler (1967, 1968, 1970) found to be optimal for people to use to organize large amounts of information. MENUNIX organizes programs into a hierarchy in which commands with similar functions are grouped together into _workbenches_. A workbench is a set of program tools especially suited for work on a specific task. Each workbench displays a cognitively manageable number of options (Miller, 1956), all in the semantic category defined by the name of the workbench. The menu display allows users to _recognize_ options instead of the more difficult process of _recalling_ them especially for many alternatives (e.g., hundreds of programs) (Slamecka, 1967), and provides a structured procedure for discovery of options, too. Programs and workbenches that serve multiple purposes are multiply represented in the program hierarchy. This is to allow for the possibility that what is apparently an ideal decomposition of the programs for one person, according to what Rosch (1973) calls _natural categories_, might not be optimal for another, possibly due to different amounts of expert knowledge in particular domains. For example, a program to sort the lines in a file could have uses for preparing reports for a teacher, for making a list of address labels for a secretary, or for a psychologist to put some data in order.

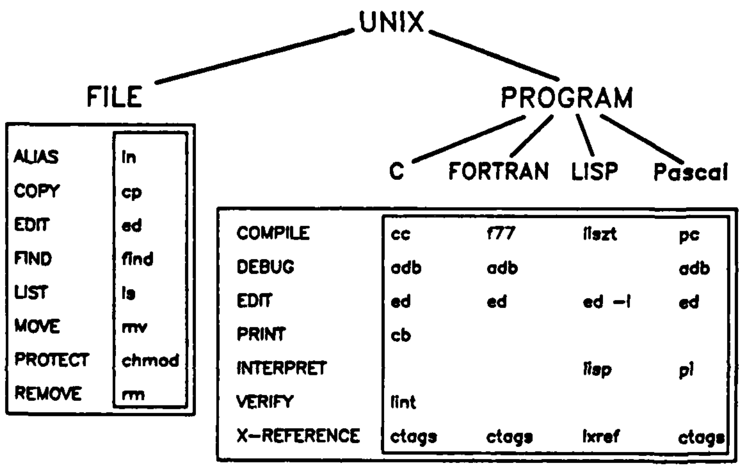
Now I illustrate how this hierarchy can be used to find programs and then I describe how the procedure of finding a program in the hierarchy can be used to name it. When users start using MENUNIX, they are presented with a high level workbench with sub-workbenches for specific tasks such as writing papers, sending and receiving electronic mail, programming, computing statistics, and so on. This is shown in Figure 13.

The user makes an initial decision about the task the program is supposed to accomplish. This is done by pressing the letter next to the name of the workbench. This letter is an easily learned mnemonic symbol, selected based on the results of Experiment 2 on optimal name-symbol pairings. From these, a new user can look at all and only those commands related to a specific task by entering the workbench for that task. Usually a workbench contains no more than ten programs. On entering a task specific workbench, there might be more specific workbenches mixed with programs. For example, in the programming workbench there are sub-workbenches for specific programming languages: LISP, FORTRAN, Pascal, etc, which can be selected with the obvious mnemonics, "l," "f," and "p," respectively. The series of mnemonic letters used to select a command can serve as an alternative name for it. See the series of PROGRAM MENU selections in Figure 14.

Such a naming scheme has several desirable properties, because programs are named according to a system of naming rules. Similar programs have similar names. This tells users that any program that operates on mail begins with "m" while those related to programming begin with "p." If a user knows about one command, the name of a related command can be inferred analogically. For example, knowing that the program to verify a FORTRAN program is called "pfv" the selecting sequence through "programming," "fortran," and "verify," it is a simple generalization to infer that the Pascal program verifier is called "ppv," and the one for LISP is called "plv." Figure 15 shows part of the MENUNIX hierarchy of programs For example, the program to move files is reached by typing "fm," and the one to copy files is called "fc," while the LISP editor is reached with "ple." This naming scheme maintains the component structure of meaning (the purpose of the program) shown by Pate (1982) to be useful for learning command names, and that related programs have related names, unlike the true UNIX names for the programs, which follow no known conventions. A program that can many uses can get several aliases, to help people find programs when and where they need them. Some potential program names, generated by analogy with existing ones, may not have existing programs (there are not C or FORTRAN interpreters). This emphasizes an important issue, that of adopting naming conventions as part of a notational system, rather than choosing what seems to be the best name at the time. Norman (1981) has complained that there is no system for naming commands on the UNIX operating system, a system where programs originate from many computer science sites, each with its own system, or lack thereof, of naming commands. Consider the names on our system of the different program compilers: Pascal's is called pc, FORTRAN's is called 77, and LISP's is called liszt. Each name--perhaps not the last--can be justified, but together, they cannot. The inconsistent naming conventions result in the programs being hard to find, hard to remember, and with little suggestiveness of their purposes. Having a system for naming commands allows programs to have names waiting for programs that have yet to be created.

The results of Experiment 1 tell us the that mnemonic symbols for workbench and program names make it relatively easy to recall the names from the symbols. After the first letter in a selection sequence name (e.g., "p" in "plc," the programming language LISP's compiler) is decoded to a name ("programming"), the results of Experiment 3 suggest that concepts related to programming are primed. This is further supported by semantic priming experiments reported by Collins and Loftus (1975). Thus, Experiments 1 and 3 predict the process of decoding a

Figure 15: Part of the MENUNIX Program Hierarchy



ALIAS	ln
COPY	cp
EDIT	ed
FIND	find
LIST	ls
MOVE	mv
PROTECT	chmod
REMOVE	rm

	C	FORTRAN	LISP	Pascal
COMPILE	cc	f77	liszt	pc
DEBUG	adb	adb		adb
EDIT	ed	ed	ed -l	ed
PRINT	cb			
INTERPRET			lisp	pi
VERIFY	lint			
X-REFERENCE	ctags	ctags	lxref	ctags

Shown here is part of the MENUNIX hierarchy of programs. Inside the high level UNIX workbench are two task specific workbenches, one for working with files, the other for programs. The workbench of programming tools contains four language specific workbenches: C, FORTRAN, LISP, and Pascal. In the FILE workbench, the true UNIX names for programs are listed in lower case in the inner box, preceded by the tasks they are used for, in upper case. A similar scheme is used for the PROGRAM workbench.

selection sequence name into its constituent selections, and its meaning.

CONCLUSIONS

Summary of Experiments and Evaluation of the Model

The interactive model of processing linguistically mediated artificial language has not been implemented into a working simulation. It does however provide a framework in which the results of the experiments described can be interpreted. In fact, the model provided the framework that motivated the hypotheses the experiments were designed to address. The experimental evidence presented in previous sections shows that the choice of symbols for names, and the choice of names for concepts, are important if the artificial language they are used in is to be a natural one. Different symbols for a name can have different association strengths to it, both in comprehending artificial language, as was shown in Experiment 1, and in generating artificial language, as was shown in Experiment 2. Different names chosen for a concept have different facilitory effects on how easily people can associate names with nearby symbols, as was shown in Experiment 3. This effect is probably due to the priming of concepts related to the chosen name that in turn prime the names paired with the nearby symbols. Taken together, the results support a multi-level model in which symbols are associated with names, and names are associated with concepts, which have semantic associations.

The size of the experimental effects should not be taken lightly. For associating symbols with names, the range of the effects from best to worst pairings was between one and two seconds <u>per symbol</u>. For associating names with concepts, the effect of a compatible name over an incompatible one was about 200 msec, but again, this effect is applied to each <u>symbol</u> being interpreted in the context of a name. In a relatively complex expression with ten symbols, the <u>difference</u> between the very best chosen notation and the very worst could be from ten to twenty seconds. At this size, the limits of human short term memory span are being tested (Peterson and Peterson, 1959) rather than the understanding of the meaning of the symbcls.

Future Work

Analog Artificial Languages

I have not discussed the use of analog representations for concepts. In an analog representation, the symbols, or the spatial relations among them, have properties in common with their meaning. For example, in international road signs, the symbol for a winding road is a winding line on a yellow sign. The shape of the symbol is an analog of the shape of the road. The color of the sign is conventionally used to denote warning of a possible hazard.

The use of analog artificial language has become more important as the technology for expressing it has advanced. It is only beginning around 1980 that computer graphics terminals along with the software for controlling them has become generally available, mainly due to advances in hardware design. For example, the XEROX Star personal computer (Smith et al, 1982) uses graphical symbols to display objects in a user's working environment. A file is shown on the user's screen as a little picture of a file folder with the file's name on it. Even operations are done graphically. To send some electronic mail, a user points to a file, "picks it up," and places it in a tray labelled "mail." Printing and duplicating files are handled similarly.

In addition to analog symbols, other features can be used to convey meaning. Size, shape, color, case, font, and slant are but some visual dimensions on which printed text can vary, and it is an important practical question of whether these attributes can be used to convey meaning, either independently or in conjunction. It is not clear how many dimensions people can integrate independently. Research on multidimensional scaling (Shepard, 1974; Wish & Carroll, 1974) has shown few classes of objects where more than three dimensions were needed to describe members of the classes, and few psychological studies of perceptual integration (Anderson, 1974) attempt experimental designs with more than three or four dimensions.

Color can be used to enhance some parts of a picture, and the meaning of color is a type of artificial language. For example, one well known code relates color and outcomes of actions, based on the red-yellow-green traffic lights on roads. Red may indicate an incorrect outcome where progress must stop, yellow may indicate a dangerous outcome where progress can be made with caution, and green may indicate a positive outcome where progress can continue. Color and brightness can also be used as a means of plotting variables by having either correspond to the magnitude of a variable. Though using color may seem uniformly advantageous, in some cases, color may be used ineffectively and inhibit efficient communication of ideas. Durrett and Trezona (1982) discuss some methods for effective use, but their presentation is based on a limited understanding of the human visual system.

High Level Constructs

In this paper, I have concentrated on artificial languages in which ideas are expressed by choosing names for concepts and symbols are chosen for names, a class of languages I call linguistically mediated artificial languages. Most of my discussion, and the experiments I presented, focus on those two stages of linguistic mediation: (1) choosing names for concepts and (2) choosing symbols for names. I touched on some higher level aspects of artificial languages such as the importance of consistency of syntax to the naturalness of an artificial language without experimental support, though from practical experience, it is clear that inconsistency causes problems. I did not discuss perhaps the most important design question of artificial languages: When is the development of an artificial language for a particular domain in order, and given that one is to be designed, what primitive conceptual units (operations and terms) should be included? I have discussed how to choose symbols for names, and even names for concepts, but not _which_ concepts get names. Finseth (1982) discusses what primitives to put into a text editing program, an artificial language designed for manipulating text interactively. What are offered are not principles of design, but properties an expert thought useful after experience with many such systems. Practical experience is often a useful basis for design, but it should not be the only basis for design decisions. Norman (1982) has stressed the need for a _cognitive engineering_ to develop principles of human-machine interaction that will guide the design of systems people can use efficiently. Perhaps future investigations on the conceptual units needed to make a natural artificial language will provide such design principles, so that design can be guided by theory and not trial and error.

Applicability of Cognitive Psychology to Design

The experiments described were designed to answer questions about how people process linguistically mediated artificial language and how to design natural artificial languages based on this knowledge, but they were also motivated by practical experience with computer program user interfaces. The interaction of theory and application is an important one. Most basic experiments on the psychology of learning and memory in the literature were designed to answer general theoretical questions, but that does not detract from their applicability to specific problems. Conversely, designing experiments to answer practical questions can do more than just answer specific questions. If care is taken, they can have theoretical implications about information processing in general. Experiments designed to answer theoretical questions while maintaining practical applicability have the added attraction that they will tend to be ecologically valid (the experiments study something people really do).

In earlier sections, I discussed the application of cognitive psychology to the design of user interfaces for computer programs. In recent years, this has become an area of intense research (Card, Moran, & Newell, 1982) because of the dramatic surge of the use of computers in society. In discussing the design of the MENUNIX interface (Perlman, 1981), I showed how processes in understanding and generating artificial language were a good framework for understanding low level aspects of human-computer interaction in addition to more obvious domains of application, such as mathematics education.

In the design of MENUNIX, psychological theory drove the design of the major components of the system as a whole. The advantage of recognition over recall (Slamecka, 1967) prompted the use of menu displays. Each menu displayed a cognitively manageable number of options (Miller, 1956). Since people organize information hierarchically by meaning (Mandler, 1967, 1968, 1970), the hundreds of programs of the UNIX operating system displayed by MENUNIX were decomposed into cognitively manageably sized menus in which task related programs were clustered. To allow for the possibility that different people might have different natural categories (Rosch, 1973), options that may fit in many categories were multiply represented in this hierarchy.

Experimental results on how to choose symbols for names and names for concepts were also applied to the design of MENUNIX. Experiment 2 showed that mnemonic letter symbols for names resulted in the fastest performance to type a symbol when presented a name (this is an abstract menu selection task), but also that non-mnemonic letter symbols produced worse performance compared to the neutral number symbols. These results were applied to the design of MENUNIX. Mnemonic letters were used to select programs from the hierarchical menu of programs, a choice possible because the programs in the hierarchy seldom changed and permanent mnemonics could be chosen. Sorted number selectors were used in the menu of files because the names and number of files changed relatively often, so choosing mnemonic letter symbols for them was not possible. Neutral number selectors were used instead. Having disjoint sets of selectors for programs and menus has the added advantage that mode errors (Norman, 1980) could be avoided. The combined results of Experiments 1 and 3 explain the process of naming programs by the sequence of mnemonic letter selectors used in accessing them via the program hierarchy.

Based on the experiments reported and their application to the design of MENUNIX, it should be clear that there is no set of guidelines that can be followed that will insure an artificial language to be a natural one. For example, letter symbols were both the best and the worst symbols for names, and _a priori_ it is often not possible to tell which they will be. Another example is that specific names can be the best and worst names for concepts, and it will depend on the individual user whether this will be the case. In many cases, using a neutral symbol or a gene·al name may the best overall solution to a general problem, though not the optimal solution to the problem under relatively constrained conditions.

Relation to Learning Mathematics

When I tell people I study how people learn mathematics, I am often told that they would like to know why people do not learn mathematics. People say they learned a complex task like their mother tongue "effortlessly," but that mathematics is beyond them. First, I claim people do not learn their first natural language effortlessly. It takes most people about twenty years of daily practice to speak as an adult. Assuming as little as seven hours of practice a day during those twenty years, by the age of twenty, a person will have had over 50,000 hours of speaking, listening to, reading, and writing a language. This is not to say that tens of thousands of hours of experience with mathematical notation is necessary to become proficient in it. Mathematical notation is much simpler than natural language. But I think it is instructive to note that what many people think of as an effortless task, is one that takes years to perfect.

The results of the experiments reported here point to another difficulty in understanding notation. The choices of symbols for names, and names for concepts, if poorly made, can increase the time to comprehend an expression to a duration longer than human short term memory span. Peterson and Peterson (1959) showed the time course of our ability to retain information over short periods of time. I hesitate to extrapolate from their experimental materials to short term memory for expressions in mathematical notation, or any artificial language, but suppose for the sake of argument that it is about fifteen seconds for seven items. Imagine a person reading some expression of mathematics that has several symbols in it. Each of these symbols has to be associated with the name they represent, and the speed of all of these associations depends on the suggestiveness of the names around them. Poorly chosen notation can result in times close to twice that of well chosen notation, as shown in Experiments 1 and 3, and this does not include such factors as syntax and the conceptual design of the part of artificial language. The point is that a person who reads an expression written with good notation might take ten seconds to read an expression while one reading the same concepts written in bad notation might take twenty seconds, due purely because of poor choices of symbols and names. At twenty seconds, I claim, understanding the concepts in the expression is more a test of short term memory than one of conceptual knowledge. For more complicated expressions, the deficit due to poorly chosen symbols and names becomes even greater because short term memory limits are reached even sooner in comprehending expressions. This effect, considering the effects of practice in Experiments 1 and 3, do not go away with moderate practice, an amount of practice probably greater than most people would get when being introduced to some mathematical concept. I would expect that higher level aspects of notation could affect processing to a much greater extent. So, one answer to the question is: People can't learn math because often they can't get through the description of a problem and remember what the problem was. Perhaps after such a failed effort a person will try again, but then again, maybe not.

REFERENCES

Akhmanova, O. Optimization of natural communication systems. The Hague: Mouton. 1977.

Anderson, N. H. Information integration theory: A brief survey. In D. H. Krantz, R. C. Atkinson, R. D. Luce, & P. Suppes (Eds.), Contemporary developments in mathematical psychology (Volume II) Measurement, psychophysics, and neural information processing. San Francisco: W. H. Freeman, 1974. pp. 236-305.

Baggett, P. Comprehension of verbal/pictorial instructions. Boulder, Colorado: Department of Psychology, University of Colorado (Boulder). September 9, 1980. (Office of Naval Research Quarterly Report.)

Barnard, P., Hammond, N., MacLean, A., & Morton, J. Learning and remembering interactive commands. In Proceedings of the human factors in computer systems meeting. Gaithesburg, Maryland: Association for Computing Machinery, 1982.

Black, J. B., & Moran, T. P. Learning and remembering command names. In Proceedings of the human factors in computer systems meeting. Gaithesburg, Maryland: Association for Computing Machinery, 1982.

Bott, R. A. A study of complex learning: Theory and methodologies. La Jolla, California: Center for Human Information Processing, University of California, San Diego. 1979.

Card, S., Moran, T., & Newell, A. Applied information-processing: The human-computer interface. Hillsdale, New Jersey: Lawrence Erlbaum Associates. 1982.

Clark, H. H., & Clark, E. V. Psychology and language: An introduction to psycholinguistics. New York: Harcourt Brace Jovanovitch. 1977.

Collins, A. M., & Loftus, E. F. A spreading activation model of semantic processing. Psychological Review, 1975, 82:6, 407-428.

Durrett, J., & Trezona, J. How to use color displays effectively. A look at the elements of color vision and their implications for programmers. BYTE, 1982, 7:4, 50-53.

Ehrenreich, S., & Moses, F. An algorithm for generating abbreviations to be used in user-computer transactions. In Proceeding of the Easter Psychological Association. 1981.

Finseth, C. A. Managing words: What capabilities should you have with a text editor?. BYTE, 1982, 7:4, 302-310.

Guralnik, D. B. (Ed.) Webster's new world dictionary. (2nd Edition). New York: William Collins + World Publishing. 1978.

Halasz, F., & Moran, T. P. Analogy considered harmful. In Proceedings of the human factors in computer systems meeting. Gaithesburg, Maryland: Association for Computing Machinery, 1982.

Hirsh-Pasek, K., Nudelman, S., & Schneider, M. L. An experimental evaluation of abbreviation schemes in limited lexicons. Blue Bell, Pennsylvania: Sperry Univac. 1982.

Hodge, M., & Pennington, F. M. Some studies of word abbreviation behavior. Journal of Experimental Psychology, 1973, 98:2, 350-361.

Hopcroft, J. E., & Ullman, J. D. Introduction to automata theory, languages and computation. Reading, Massachusetts: Addison-Wesley. 1979.

Kucera, H., & Francis, W. N. Computational analysis of present day American English. Providence, Rhode Island: Brown University Press. 1967.

Mandler, G. Organization and memory. In K. W. Spence, & J. T. Spence (Eds.), The psychology of Learning and motivation: Advances in research and theory. New York: Academic Press, 1967.

Mandler, G. Association and organization: Facts, fancies and theories. In T. R. Dixon, & D. L. Horton (Eds.), Verbal behavior and general behavior theory. Englewood Cliffs, New Jersey: Prentice Hall, 1968.

Mandler, G. Words, lists, and categories: An experimental view of organized memory. In J. L. Cowan (Ed.), Studies in thought and language. Tucson, AZ: University of Arizona Press, 1970.

McClelland, J. L., & Rumelhart, D. E. An interactive activation model of context effects in letter perception: Part 1. An account of basic findings. Psychological Review, 1981, 88:5, 375-407.

Miller, G. A. The magical number seven, plus or minus two: Some limits on our capacity for processing information. Psychological Review, 1956, 63, 81-97.

Moran, T. P. The command language grammar: A representation for the user interface of int⁻ractive computer systems. International Journal of Man-Machine Studies, 1981, 15, 3-50.

Neely, J. H. Semantic priming and retrieval from lexical memory: Roles of Inhibitionless spreading activation and limited-capacity attention. Journal of Experimental Psychology: General, 1977, 106:3, 226-254.

Norman, D. A. Categorization of action slips. Psychologicai Review, 1981, 88, 1-15.

Norman, D. A. The trouble with UNIX. Datamation, 1981, 27, 138-150.

Norman, D. A. Steps toward a cognitive engineering: Design rules based on analyses of human error. In Proceedings of the human factors in computer systems meeting. Gaithesburg, Maryland: Association for Computing Machinery, 1982.

Norman, D. A., & Rumelhart, D. E. (Eds.) Explorations in Cognition. San Francisco: W. H. Freeman. 1975.

Paivio, A. Mental imagery in associative learning and memory. Psychological Review, 1969, 76, 241-263.

Pate, D. S. Lexical recomposition: Contextual influences on verb specificity. La Jolla, California: University of California, San Diego, 1982. (Unpublished doctoral dissertation.)

Perlman, G. Data analysis programs for the UNIX operating system. Behavior Research Methods and Instrumentation, 1980, 12:5, 554-558.

Perlman, G. Making mathematical notation more meaningful. The Mathematics Teacher, 1982, 75:6, 462-466.

Perlman, G. Two papers in cognitive engineering: The design of an interface to a programming system, and MENUNIX: A menu-based interface to UNIX (User manual). La Jolla, California: Center for Human Information Processing, University of California, San Diego. November, 1981. (Report No. 8105.)

Peterson, L. R., & Peterson, M. J. Short-term retention of verbal items. Journal of Experimental Psychology, 1959, 58, 193-198.

Pratt, T. W. Programming languages: Design and implementation. Englewood Cliffs, New Jersey: Prentice-Hall. 1975.

Reber, A. S. Implicit learning of artificial grammars. Journal of Verbal Learning and Verbal Behavior, 1967, 6, 855-863.

Reisner, P. Formal grammar and human factors design of an interactive graphics system. IEEE Transactions on Software Engineering, 1981, 7, 229-240.

Richie, D. M., & Thompson, K. The UNIX time-sharing system. Communications of the ACM, 1974, 17, 365-375.

Richie, D. M., & Thompson, K. The UNIX time-sharing system. Bell System Technical Journal, 1978, 57, 1905-1929.

Rosch, E. Cognitive representations of semantic categories. Journal of Experimental Psychology: General, 1975, 104:3, 192-233.

Rosenberg, J. Evaluating the suggestiveness of Command Names. In Proceedings of the human factors in computer systems meeting. Gaithesburg, Maryland: Association for Computing Machinery, 1982.

Rumelhart, D. E., & Norman, D. A. Accretion, tuning, and restructuring: Three modes of learning. In J. W. Cotton, & R. Klatzky (Eds.), Semantic Factors in Cognition. Hillsdale, New Jersey: Lawrence Erlbaum Associates, 1978.

Rumelhart, D. E., & Norman, D. A. Analogical processes in learning. In J. R. Anderson (Ed.), Cognitive skills and their acquisition. Hillsdale, New Jersey: Lawrence Erlbaum Associates, 1981.

Rumelhart, D. E., & McClelland, J. L. An interactive model of contexts effects in letter perception: Part 2. The contextual enhancement effect and some tests and extensions of the model. Psychological Review, 1982, 89:1, 60-94.

Savitch, W. Formal languages. La Jolla, California: University of California, San Diego. 1979.

Sebeok, T. A. The sign and its masters. Austin, Texas: University of Texas Press. 1979.

Shepard, R. N. Representation of structure in similarity data: Problems and prospects. Psychometrica, 1974, 39:4, 373-421.

Shneiderman, B. Software psychology: Human factors in computer and information systems. New York: Winthrop. 1980.

Skemp, R. R. The psych of learning mathematics. London: Penguin Books. 1971.

Slamecka, N. J. Recall and recognition in list-discrimination tasks as a functi[illegible] of alternatives. Journal of Experimental Psychology, 1967, 74, 87-192.

Smith, D. C., Irby, C., Kimball, R., Verplank, B., & Harslem, E. Designing the Star user interface. BYTE, 1974, 7:4, 242-282.

Streeter, L., Ackroff, J., & Taylor, G. On abbreviating command names. In Proceedings of the Eastern Psychological Association. 1981.

Stroop, J. R. Studies of interference in serial verbal reactions. Journal of Experimental Psychology, 1935, 18, 643-662.

Underwood, B. J. Degree of learning and the measurement of forgetting. Journal of Verbal Learning and Verbal Behavior, 1964, 3, 112-129.

Whitehead, A. N. An introduction to mathematics. 1911.

Whorf, B. L. Language, thought, and reality. Cambridge, Massachusetts: The M.I.T. Press. 1956.

Wish, M., & Carroll, J. D. Applications of individual differences scaling to studies of human perception and judgment. In Handbook of perception II: Psychophysical Judgment and Measurement. New York: Academic Press, 1974. pp. 449-491.

Cognitive Science ONR Technical Report List

8001. Donald R. Gentner, Jonathan Grudin, and Eileen Conway. Finger Movements in Transcription Typing. May 1980.

8002. James L. McClelland and David E. Rumelhart. An Interactive Activation Model of the Effect of Context in Perception: Part I. May 1980.

8003. David E. Rumelhart and James L. McClelland. An Interactive Activation Model of the Effect of Context in Perception: Part II. July 1980.

8004. Donald A. Norman. Errors in Human Performance. August 1980.

8005. David E. Rumelhart and Donald A. Norman. Analogical Processes in Learning. September 1980.

8006. Donald A. Norman and Tim Shallice. Attention to Action: Willed and Automatic Control of Behavior. December 1980.

8101. David E. Rumelhart. Understanding Understanding. January 1981.

8102. David E. Rumelhart and Donald A. Norman. Simulating a Skilled Typist: A Study of Skilled Cognitive-Motor Performance. May 1981.

8103. Donald R. Gentner. Skilled Finger Movements in Typing. July 1981.

8104. Michael I. Jordan. The Timing of Endpoints in [illegible]ment. November 1981.

8105. Gary Perlman. Two Papers in Cognitive Engineering: The Design of an Interface to a Programming System and MENUNIX: A Menu-Based Interface to UNIX (User Manual). November 1981.

8106. Donald A. Norman and Diane Fisher. Why Alphabetic Keyboards Are Not Easy to Use: Keyboard Layout Doesn't Much Matter. November 1981.

8107. Donald R. Gentner. Evidence Against a Central Control Model of Timing in Typing. December 1981.

8201. Jonathan T. Grudin and Serge Larochelle. Digraph Frequency Effects in Skilled Typing. February 1982.

8202. Jonathan T. Grudin. Central Control of Timing in Skilled Typing. February 1982.

8203. Amy Geoffroy and Donald A. Norman. Ease of Tapping the Fingers in a Sequence Depends on the Mental Encoding. March 1982.

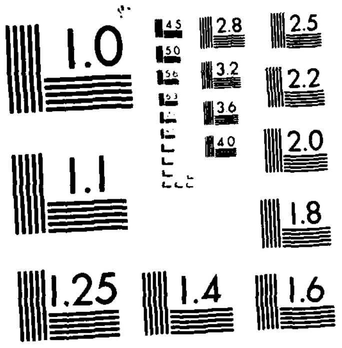
8204. LNR Research Group. Studies of Typing from the LNR Research Group: The role of context, differences in skill level, errors, hand movements, and a computer simulation. May 1982.

8205. Don Norman. Five Papers on Human-Machine Interaction. May 1982.

8206. Naomi Miyake. Constructive Interaction. June 1982.

8207. Donald R. Gentner. The Development of Typewriting Skill. September 1982.

8208. Gary Perlman. Natural Artificial Languages: Low-Level Processes. December 1982.



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963 A

Navy

1 Robert Ahlers
Code N711
Human Factors Laboratory
NAVTRAEQUIPCEN
Orlando, FL 32813

1 CDR Robert J. Biersner
Naval Medical R&D Command
National Naval Medical Center
Bethesda, MD 20814

1 Dr. Robert Blanchard
Navy Personnel R&D Center
San Diego, CA 92152

1 Liaison Scientist
Office of Naval Research
Branch Office, London
Box 39
FPO New York, NY 09510

1 Dr. Richard Cantone
Navy Research Laboratory
Code 7510
Washington, DC 20375

1 Chief of Naval Education and Training
Liason Office
Air Force Human Resource Laboratory
Operations Training Division
WILLIAMS AFB, AZ 85224

1 Dr. Stanley Collyer
Office of Naval Technology
800 N. Quincy Street
Arlington, VA 22217

1 CDR Mike Curran
Office of Naval Research
800 N. Quincy St.
Code 270
Arlington, VA 22217

1 Dr. Tom Duffy
Navy Personnel R&D Center
San Diego, CA 92152

1 Dr. Carl E. Englund
Naval Health Research Center
Code 8060 Environmental Physiology Dept
P.O. Box 85122
San Diego, CA 92138

Navy

1 DR. PAT FEDERICO
Code P13
NPRDC
San Diego, CA 92152

1 Dr. John Ford
Navy Personnel R&D Center
San Diego, CA 92152

1 Dr. Jude Franklin
Code 7510
Navy Research Laboratory
Washington, DC 20375

1 Dr. Mike Gaynor
Navy Research Laboratory
Code 7510
Washington, DC 20375

1 LT Steven D. Harris, MSC, USN
RFD 1, Box 243
Riner, VA 24149

1 Dr. Jim Hollan
Code 304
Navy Personnel R & D Center
San Diego, CA 92152

1 Dr. Ed Hutchins
Navy Personnel R&D Center
San Diego, CA 92152

1 Dr. Norman J. Kerr
Chief of Naval Technical Training
Naval Air Station Memphis (75)
Millington, TN 38054

1 Dr. Peter Kincaid
Training Analysis & Evaluation Group
Dept. of the Navy
Orlando, FL 32813

1 Dr. James Lester
ONR Detachment
495 Summer Street
Boston, MA 02210

1 Dr. William L. Maloy
Principal Civilian Advisor for
Education and Training
Naval Training Command, Code 00A
Pensacola, FL 32508

Navy

1 CAPT Richard L. Martin, USN
Commanding Officer
USS Carl Vinson (CVN-70)
FPO New York, NY 09558

1 Dr. George Moeller
Director, Behavioral Sciences Dept.
Naval Submarine Medical Research Lab
Naval Submarine Base
Groton, CT 63409

1 Dr William Montague
NPRDC Code 13
San Diego, CA 92152

1 Technical Director
Navy Personnel R&D Center
San Diego, CA 92152

6 Commanding Officer
Naval Research Laboratory
Code 2627
Washington, DC 20390

1 Office of Naval Research
Code 433
800 N. Quincy SStreet
Arlington, VA 22217

6 Personnel & Training Research Group
Code 442PT
Office of Naval Research
Arlington, VA 22217

1 Psychologist
ONR Branch Office
1030 East Green Street
Pasadena, CA 91101

1 Office of the Chief of Naval Operations
Research Development & Studies Branch
OP 115
Washington, DC 20350

1 Dr. Gary Poock
Operations Research Department
Code 55PK
Naval Postgraduate School
Monterey, CA 93940

1 Dr. Gil Ricard
Code N711
NTEC
Orlando, FL 32813

Navy

1 Dr. Worth Scanland
CNET (N-5)
NAS, Pensacola, FL 32508

1 Mr. Irving Schiff
Dept. of the Navy
Chief of Naval Operations
OP 113
Washington, DC 20350

1 Dr. Robert G. Smith
Office of Chief of Naval Operations
OP-987H
Washington, DC 20350

1 Dr. Alfred F. Smode, Director
Training Analysis & Evaluation Group
Dept. of the Navy
Orlando, FL 32813

1 Dr. Richard Sorensen
Navy Personnel R&D Center
San Diego, CA 92152

1 Dr. Frederick Steinheiser
CNO - OP115
Navy Annex
Arlington, VA 20370

1 Roger Weissinger-Baylon
Department of Administrative Sciences
Naval Postgraduate School
Monterey, CA 93940

1 Dr. Douglas Wetzel
Code 12
Navy Personnel R&D Center
San Diego, CA 92152

1 Mr John H. Wolfe
Navy Personnel R&D Center
San Diego, CA 92152

Marine Corps

1 H. William Greenup
Education Advisor (E031)
Education Center, MCDEC
Quantico, VA 22134

1 Special Assistant for Marine
Corps Matters
Code 100M
Office of Naval Research
300 N. Quincy St.
Arlington, VA 22217

1 DR. A.L. SLAFKOSKY
SCIENTIFIC ADVISOR (CODE RD-1)
HQ, U.S. MARINE CORPS
WASHINGTON, DC 20380

Army

1 Technical Director
U. S. Army Research Institute for the
Behavioral and Social Sciences
5001 Eisenhower Avenue
Alexandria, VA 22333

1 Mr. James Baker
Army Research Institute
5001 Eisenhower Avenue
Alexandria, VA 22333

1 Dr. Beatrice J. Farr
U. S. Army Research Institute
5001 Eisenhower Avenue
Alexandria, VA 22333

1 Dr. Milton S. Katz
Training Technical Area
U.S. Army Research Institute
5001 Eisenhower Avenue
Alexandria, VA 22333

1 Dr. Marshall Narva
US Army Research Institute for the
Behavioral & Social Sciences
5001 Eisenhower Avenue
Alexandria, VA 22333

1 Dr. Harold F. O'Neil, Jr.
Director, Training Research Lab
Army Research Institute
5001 Eisenhower Avenue
Alexandria, VA 22333

1 Dr. Joseph Psotka
Army Research Institute
5001 Eisenhower Avenue
Alexandria, VA 22333

1 Dr. Robert Sasmor
U. S. Army Research Institute for the
Behavioral and Social Sciences
5001 Eisenhower Avenue
Alexandria, VA 22333

1 Dr. Robert Wisher
Army Research Institute
5001 Eisenhower Avenue
Alexandria, VA 22333

Air Force

1 AFHRL/LRS
Attn: Susan Ewing
WPAFB
WPAFB, OH 45433

1 U.S. Air Force Office of Scientific
Research
Life Sciences Directorate, NL
Bolling Air Force Base
Washington, DC 20332

1 Air University Library
AUL/LSE 76/443
Maxwell AFB, AL 36112

1 Dr. Earl A. Alluisi
HQ, AFHRL (AFSC)
Brooks AFB, TX 78235

1 Bryan Dallman
AFHRL/LRT
Lowry AFB, CO 80230

1 Dr. Alfred R. Fregly
AFOSR/NL
Bolling AFB, DC 20332

1 Dr. Genevieve Haddad
Program Manager
Life Sciences Directorate
AFOSR
Bolling AFB, DC 20332

1 Dr. T. M. Longridge
AFHRL/OTGT
Williams AFB, AZ 85224

1 Dr. Joseph Yasatuke
AFHRL/OT
Williams AFB, AZ 58224

Department of Defense

12 Defense Technical Information Center
Cameron Station, Bldg 5
Alexandria, VA 22314
Attn: TC

1 Military Assistant for Training and
Personnel Technology
Office of the Under Secretary of Defens
for Research & Engineering
Room 3D129, The Pentagon
Washington, DC 20301

1 Major Jack Thorpe
DARPA
1400 Wilson Blvd.
Arlington, VA 22209

Civilian Agencies

1 Dr. Patricia A. Butler
NIE-BRN Bldg, Stop # 7
1200 19th St., NW
Washington, DC 20208

1 Dr. Susan Chipman
Learning and Development
National Institute of Education
1200 19th Street NW
Washington, DC 20208

1 Dr. Andrew R. Molnar
Office of Scientific and Engineering
Personnel and Education
National Science Foundation
Washington, DC 20550

1 Chief, Psychological Reserch Branch
U. S. Coast Guard (G-P-1/2/TP42)
Washington, DC 20593

1 Dr. Edward C. Weiss
National Science Foundation
1800 G Street, NW
Washington, DC 20550

1 Dr. Frank Withrow
U. S. Office of Education
400 Maryland Ave. SW
Washington, DC 20202

1 Dr. Joseph L. Young, Director
Memory & Cognitive Processes
National Science Foundation
Washington, DC 20550

Private Sector

1 Dr. John R. Anderson
Department of Psychology
Carnegie-Mellon University
Pittsburgh, PA 15213

1 Dr. John Annett
Department of Psychology
University of Warwick
Coventry CV4 7AJ
ENGLAND

1 Dr. Michael Atwood
Bell Laboratories
11900 North Pecos St.
Denver, CO 80234

1 Psychological Research Unit
Dept. of Defense (Army Office)
Campbell Park Offices
Canberra ACT 2600
AUSTRALIA

1 Dr. Alan Baddeley
Medical Research Council
Applied Psychology Unit
15 Chaucer Road
Cambridge CB2 2EF
ENGLAND

1 Dr. Patricia Baggett
Department of Psychology
University of Colorado
Boulder, CO 80309

1 Ms. Carole A. Bagley
Minnesota Educational Computing
Consortium
2354 Hidden Valley Lane
Stillwater, MN 55082

1 Dr. Jonathan Baron
80 Glenn Avenue
Berwyn, PA 19312

1 Mr. Avron Barr
Department of Computer Science
Stanford University
Stanford, CA 94305

1 Dr. Menucha Birenbaum
School of Education
Tel Aviv University
Tel Aviv, Ramat Aviv 69978
Israel

Private Sector

1 Dr. Werner Birke
DezWPs im Streitkraefteamt
Postfach 20 50 03
D-5300 Bonn 2
WEST GERMANY

1 Dr. John Black
Yale University
Box 11A, Yale Station
New Haven, CT 06520

1 Dr. Lyle Bourne
Department of Psychology
University of Colorado
Boulder, CO 80309

1 Dr. John S. Brown
XEROX Palo Alto Research Center
3333 Coyote Road
Palo Alto, CA 94304

1 Bundministerium der Verteidigung
-Referat P II 4-
Psychological Service
Postfach 1328
D-5300 Bonn 1
F. R. of Germany

1 Dr. Jaime Carbonell
Carnegie-Mellon University
Department of Psychology
Pittsburgh, PA 15213

1 Dr. Pat Carpenter
Department of Psychology
Carnegie-Mellon University
Pittsburgh, PA 15213

1 Dr. William Chase
Department of Psychology
Carnegie Mellon University
Pittsburgh, PA 15213

1 Dr. Micheline Chi
Learning R & D Center
University of Pittsburgh
3939 O'Hara Street
Pittsburgh, PA 15213

1 Dr. William Clancey
Department of Computer Science
Stanford University
Stanford, CA 94306

Private Sector

1 Dr. Michael Cole
University of California
at San Diego
Laboratory of Comparative
Human Cognition - D003A
La Jolla, CA 92093

1 Dr. Allan M. Collins
Bolt Beranek & Newman, Inc.
50 Moulton Street
Cambridge, MA 02138

1 Dr. Kenneth B. Cross
Anacapa Sciences, Inc.
P.O. Drawer Q
Santa Barbara, CA 93102

1 ERIC Facility-Acquisitions
4833 Rugby Avenue
Bethesda, MD 20014

1 Dr. Paul Feltovich
Department of Medical Education
Southern Illinois University
School of Medicine
P.O. Box 3926
Springfield, IL 62708

1 Professor Reuven Feuerstein
HWCRI Rehov Karmon 6
Bet Hakerem
Jerusalem
Israel

1 Mr. Wallace Feurzeig
Department of Educational Technology
Bolt Beranek & Newman
10 Moulton St.
Cambridge, MA 02238

1 Dr. Victor Fields
Dept. of Psychology
Montgomery College
Rockville, MD 20850

1 Univ. Prof. Dr. Gerhard Fischer
Liebiggasse 5/3
A 1010 Vienna
AUSTRIA

1 Dr. Dexter Fletcher
WICAT Research Institute
1875 S. State St.
Orem, UT 22333

Private Sector

1 Dr. John R. Frederiksen
Bolt Beranek & Newman
50 Moulton Street
Cambridge, MA 02138

1 Dr. Michael Genesereth
Department of Computer Science
Stanford University
Stanford, CA 94305

1 Dr. Robert Glaser
Learning Research & Development Center
University of Pittsburgh
3939 O'Hara Street
PITTSBURGH, PA 15260

1 Dr. Josph Goguen
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025

1 Dr. Daniel Gopher
Department of Psychology
University of Illinois
Champaign, IL 61820

1 DR. JAMES G. GREENO
LRDC
UNIVERSITY OF PITTSBURGH
3939 O'HARA STREET
PITTSBURGH, PA 15213

1 Dr. Barbara Hayes-Roth
Department of Computer Science
Stanford University
Stanford, CA 95305

1 Dr. Frederick Hayes-Roth
Teknowledge
525 University Ave.
Palo Alto, CA 94301

1 Dr. Kristina Hooper
Clark Kerr Hall
University of California
Santa Cruz, CA 95060

1 Dr. Earl Hunt
Dept. of Psychology
University of Washington
Seattle, WA 98105

Private Sector

1 Dr. David Kieras
Department of Psychology
University of Arizona
Tuscon, AZ 85721

1 Dr. Walter Kintsch
Department of Psychology
University of Colorado
Boulder, CO 80302

1 Dr. Stephen Kosslyn
Department of Psychology
Brandeis University
Waltham, MA 02254

1 Dr. Pat Langley
Carnegie-Mellon University
Pittsburgh, PA 15213

1 Dr. Marcy Lansman
The L. L. Thurstone Psychometric
Laboratory
University of North Carolina
Davie Hall 013A
Chapel Hill, NC 27514

1 Dr. Jill Larkin
Department of Psychology
Carnegie Mellon University
Pittsburgh, PA 15213

1 Dr. Alan Lesgold
Learning R&D Center
University of Pittsburgh
3939 O'Hara Street
Pittsburgh, PA 15260

1 Dr. Jim Levin
University of California
at San Diego
Laboratory fof Comparative
Human Cognition - D003A
La Jolla, CA 92093

1 Dr. Michael Levine
Department of Educational Psychology
210 Education Bldg.
University of Illinois
Champaign, IL 61801

1 Dr. Jay McClelland
Department of Psychology
MIT
Cambridge, MA 02139

Private Sector

1 Dr. James R. Miller
Texas Instruments, Inc.
Central Research Laboratory
P. O. Box 226015, MS238
Dallas, TX 75266

1 Dr. Mark Miller
Computer Thought Corporation
1721 West Plane Parkway
Plano, TX 75075

1 Dr. Tom Moran
Xerox PARC
3333 Coyote Hill Road
Palo Alto, CA 94304

1 Dr. Allen Munro
Behavioral Technology Laboratories
1845 Elena Ave., Fourth Floor
Redondo Beach, CA 90277

1 Committee on Human Factors
JH 811
2101 Constitution Ave. NW
Washington, DC 20418

1 Dr. Jesse Orlansky
Institute for Defense Analyses
1801 N. Beauregard St.
Alexandria, VA 22311

1 Dr. Seymour A. Papert
Massachusetts Institute of Technology
Artificial Intelligence Lab
545 Technology Square
Cambridge, MA 02139

1 Dr. Nancy Pennington
University of Chicago
5801 S. Ellis Avenue
Chicago, IL 60637

1 Mr. L. Petrullo
2431 N. Edgewood Street
ARLINGTON, VA 22207

1 Dr. Richard A. Pollak
Director, Special Projects
Minnesota Educational Computing Consort
2520 Broadway Drive
St. Paul,MN

Private Sector

1 DR. PETER POLSON
DEPT. OF PSYCHOLOGY
UNIVERSITY OF COLORADO
BOULDER, CO 80309

1 Dr. Fred Reif
Physics Department
University of California
Berkeley, CA 94720

1 Dr. Lauren Resnick
LRDC
University of Pittsburgh
3939 O'Hara Street
Pittsburgh, PA 1521

1 Mary S. Riley
Program in Cognitive Science
Center for Human Information Processing
University of California, San Diego
La Jolla, CA 92093

1 Dr. Andrew M. Rose
American Institutes for Research
1055 Thomas Jefferson St. NW
Washington, DC 20007

1 Dr. Ernst Z. Rothkopf
Bell Laboratories
Murray Hill, NJ 07974

1 Dr. William B. Rouse
Georgia Institute of Technology
School of Industrial & Systems
Engineering
Atlanta, GA 30332

1 Dr. Michael J. Samet
Perceptronics, Inc
6271 Variel Avenue
Woodland Hills, CA 91364

1 Dr. Roger Schank
Yale University
Department of Computer Science
P.O. Box 2158
New Haven, CT 06520

1 Dr. Walter Schneider
Psychology Department
603 E. Daniel
Champaign, IL 61820

Private Sector

1 Dr. Alan Schoenfeld
Mathematics and Education
The University of Rochester
Rochester, NY 14627

1 DR. ROBERT J. SEIDEL
INSTRUCTIONAL TECHNOLOGY GROUP
HUMRRO
300 N. WASHINGTON ST.
ALEXANDRIA, VA 22314

1 Mr. Colin Sheppard
Applied Psychology Unit
Admiralty Marine Technology Est.
Teddington, Middlesex
United Kingdom

1 Dr. H. Wallace Sinaiko
Program Director
Manpower Research and Advisory Services
Smithsonian Institution
801 North Pitt Street
Alexandria, VA 22314

1 Dr. Edward E. Smith
Bolt Beranek & Newman, Inc.
50 Moulton Street
Cambridge, MA 02138

1 Dr. Eliott Soloway
Yale University
Department of Computer Science
P.O. Box 2158
New Haven, CT 06520

1 Dr. Kathryn T. Spoehr
Psychology Department
Brown University
Providence, RI 02912

1 Dr. Albert Stevens
Bolt Beranek & Newman, Inc.
10 Moulton St.
Cambridge, MA 02238

1 David E. Stone, Ph.D.
Hazeltine Corporation
7680 Old Springhouse Road
McLean, VA 22102

Private Sector

1 DR. PATRICK SUPPES
INSTITUTE FOR MATHEMATICAL STUDIES IN
THE SOCIAL SCIENCES
STANFORD UNIVERSITY
STANFORD, CA 94305

1 Dr. John Thomas
IBM Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

1 Dr. Perry W. Thorndyke
Perceptronics, Inc.
545 Middlefield Road, Suite 140
Menlo Park, CA 94025

1 Dr. Douglas Towne
Univ. of So. California
Behavioral Technology Labs
1845 S. Elena Ave.
Redondo Beach, CA 90277

1 Dr. Kurt Van Lehn
Zerox PARC
3333 Coyote Hill Road
Palo Alto, CA 94304

1 DR. GERSHON WELTMAN
PERCEPTRONICS INC.
6271 VARIEL AVE.
WOODLAND HILLS, CA 91367

1 Dr. Keith T. Wescourt
Perceptronics, Inc.
545 Middlefield Road, Suite 140
Menlo Park, CA 94025

1 William B. Whitten
Bell Laboratories
2D-610
Holmdel, NJ 07733

1 Dr. Christopher Wickens
Department of Psychology
University of Illinois
Champaign, IL 61820

1 Dr. Mike Williams
Zerox PARC
3333 Coyote Hill Road
Palo Alto, CA 94304

4-8

DTI